SANDIA REPORT

SAND2011-5702
Unlimited Release
Printed August 2011

Measuring and Tuning Energy Efficiency on Large Scale High Performance Computing Platforms

James H. Laros III

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2011-5702
Unlimited Release
Printed August 2011

Measuring and Tuning Energy Efficiency on Large Scale High Performance Computing Platforms

James H. Laros III
Scalable Systems Architectures
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-9999
jhlaros@sandia.gov

Abstract

Recognition of the importance of power in the field of High Performance Computing, whether it be as an obstacle, expense or design consideration, has never been greater and more pervasive. While research has been conducted on many related aspects, there is a stark absence of work focused on large scale High Performance Computing. Part of the reason is the lack of measurement capability on small or large platforms. Typical research is conducted using coarse methods of measurement, such as inserting a power meter between the power source and the platform, or fine grained measurements using custom instrumented boards (with obvious limitations in scale). To collect the measurements necessary to analyze real scientific computing applications, at large scale, an in-situ measurement capability must exist on a large scale capability class platform.

In response to this challenge, we exploit the unique power measurement capabilities of the Cray XT architecture to gain an understanding of the power requirements of important DOE/NNSA production scientific computing applications running at large scale (thousands of nodes), while simultaneously collecting current and voltage measurements on the hosting nodes. We examine the effects of both CPU and network bandwidth tuning and demonstrate energy savings opportunities of up to 39% with no impact on performance. Even higher energy savings are demonstrated with little performance impact. Our work is done from a system perspective rather than a focused node level perspective. Capturing any scale effects in our experimental results was a key factor.

We provide strong evidence that next generation large scale platforms must not only approach CPU frequency scaling differently but would also greatly benefit from the capability to tune other platform components to achieve energy efficient performance.

Acknowledgments

I would like to acknowledge the following people who have contributed in various ways, all of which have been significant, to the successful completion of this work: Dr. Wei Shu - Thesis Advisor and collaborator, University of New Mexico Electrical and Computer Engineering, James A. Ang - Manger, Scalable Computer Architectures, Sandia National Laboratories, Kevin Pedretti, Sue Kelly, John Vandyke, and Courtenay Vaughan - Collaborators, Technical Staff Sandia National Laboratories.

The following agencies have provided funding directly or indirectly to this research: National Nuclear Security Agency (NNSA) Advanced Simulation and Computing program (ASC) and the Department of Energy's (DOE) Innovative and Novel Computational Impact on Theory and Experiment (INSITE) program.

Contents

Appendix

List of Figures

List of Tables

This page intentionally left blank.

Chapter 1

Introduction

Motivation

There are three *walls* in CPU chip architecture design, memory, instruction level parallelism and power. Power is widely accepted as the tallest of these walls. This limitation is partially due to the exponential increase in power with each factorial increase of frequency. Additionally, power must be removed from an increasingly smaller area in the form of heat. Addressing these issues for mobile and server CPU chip architectures has long established motivations.

The High Performance Computing (HPC) community has for many years ignored power as a major factor in favor of increased performance. With annual power costs on track to rival acquisition costs of next generation platforms, power has increasingly been identified by the HPC community as the *tall pole* in the path to Exascale systems. Ubiquitous in the top three considerations of virtually every report on next generation or Exascale platforms, power has been recognized across the board by government agencies and commercial enterprises alike as possibly the greatest challenge in fielding future HPC platforms.

Existing hardware has been successfully leveraged by present day operating systems to conserve energy whenever possible but these approaches have proved ineffective and even detrimental at large scale. While hardware must provide part of the solution, how these solutions are leveraged on large scale platforms requires a new and flexible approach. It is particularly important that any approach taken has a system-level, rather than node-level, view of these issues.

Thesis Overview

In response to this challenge our research has leveraged a thus far unique ability to measure current draw and voltage, in-situ, on a large HPC platform at a very fine granularity and high frequency. As part of our research, our experiments begin with the goal of reducing power consumption during idle cycles. We extend this concept for multi-core architectures by ensuring cores not in use remain idle during application execution. It is not at all uncommon for HPC applications to use fewer than the number of available cores per node. For many scientific applications, the memory wall in the form of capacity, capability or both force users to limit the number of

cores per node to better balance their application memory requirements at scale. While conserving power during idle cycles can produce large energy and related cost savings we endeavored to explore possible energy efficiencies during application run-time on active CPU cores.

HPC applications are typically bulk synchronous. In [1] bulk synchronous programs are described to have three execution phases; computation, communication and synchronization. In [2] the authors describe how operating system noise can effectively slow overall computation of bulk synchronous programs. In short, the runtime of an application will be throttled by the slowest MPI rank involved in a bulk synchronous computation. Allowing frequency changes that are dictated locally, rather than from the systems perspective, can cause the equivalent of operating system noise (or jitter).

Our next experiment targeted modifying CPU frequencies during application execution. Initially, we believed to achieve significant savings, without unacceptable impacts in power use, frequency scaling would have to be performed during natural wait states in application execution, for example during communication phases. Our experiments have shown a less aggressive, and likely more stable, approach of static frequency scaling can produce large energy savings with little to no performance penalty.

Our final experiments take advantage of the ability to tune the performance of network components of a large scale HPC platform. We claim there exists a *sweet spot* for most if not all HPC applications run at large scale where maximum energy efficiency can be achieved without unacceptable performance trade-offs. We provide a large amount of experimental data to support this claim.

Our experiments were conducted on two Cray XT class platforms; Red Storm located at Sandia National Laboratories and Jaguar hosted by Oak Ridge National Laboratory. The results of our experiments clearly indicate that opportunities exist to save energy by tuning platform components, individually or together, while maintaining application performance. Our ultimate goal is to reduce energy consumption of real applications run at very large scale while minimizing the impact on run-time performance (defined for our purposes as wall-clock execution time).

Evaluating acceptable trade-offs between energy efficiency and run-time performance is, of course, somewhat subjective. Our work indicates that the parameters of these trade-offs are application dependent. While the HPC community has traditionally prioritized performance above all metrics, future per-processor or per-platform power requirements will likely alter our priorities and place more importance on energy efficiency metrics like FLOPS/Watt or Energy Delay Product (EDP). While we are highly motivated to explore power savings wherever they may be found, performance remains a critical parameter of our evaluation.

Related Work

Power, as it relates to computers and computation, has been researched from many perspectives. Certainly the largest body of work has been done in collaboration by Ge, Feng and Cameron

(in some cases in association with other researchers). In [3] and [4] the authors introduce PowerPack. PowerPack is a framework for profiling and analyzing scientific applications on distributed systems. The authors frame the problem well warning of future costs and stressing the need for component level analysis. We completely agree. In their work the importance of fine granularity component level measurement is recognized. While it was clear to us that direct measurements are taken at node level, it is less clear that some individual components were directly measured. It is clear, however, that much effort was taken to experimentally isolate the power draw for individual components from the overall nodal measurement and the frequency of the measurements were good. While this work does rely on a direct measurement technique only one node, of a 32 node cluster, was instrumented. They use a technique called "node remapping" in which they emulate measuring a single application run by executing an application that runs on M nodes M number of times ensuring that node assignments are rotated for each execution. There are obvious scalability limitations to this technique but their analysis techniques are rigorous and their results valuable at small scale for similar clusters built with commodity components. While we would like to conclude, as the authors do, that power has a direct relationship on reliability as stated by Arrhenius' Law there is evidence to the contrary in an extensive study done at Google [5]. This work shows there is not a strong correlation between temperature and reliability of disks. Intuitively, it would seem that components with moving parts would be more likely affected by temperature than a CPU, for example. Hsu in [6], however, claims informal empirical data supports Arrenhius' equation as applied to cluster systems. We feel this remains an open question but certainly lowering power and therefore heat seems unlikely to increase failure rates based on currently published research. At a minimum there is a resulting savings in heat removal. We will not make claims that our work will benefit reliability until we have supporting data but will consider this a potential benefit of this line of research.

In [7] the authors improve on the single node sampling technique described in [3] and [4] by expanding their collection capability to 16 nodes (this seems to remain the limit on node count in the remaining publications). The NEMO power aware cluster is comprised of laptops which allow the necessary measurements to be taken. These measurements are supplemented and verified by contrasting them with readings obtained from an external power monitoring device. On the downside, it seems that total node power is the granularity for this cluster unless they additionally instrument a single node as described in [3] and extrapolate. The frequency of the power samples drops from four per second to one sample every 15 to 20 seconds. Regardless, there configuration provides good small scale information. The authors nicely define three strategies of scheduling using Dynamic Voltage Scaling (DVS). The approach taken in our research equates to what they define as *external*, scheduling from the command line. They also evaluate what we would call *automatic* (scheduling accomplished by a daemon such as the cpuspeed daemon) and *internal* (scheduling initiated by the application). Our future work will be done using a form similar to internal scheduling.

Another important aspect of the approach taken by these researchers, in much of their work, is their attempts to provide a useful single or fused metric to gage success. Energy Delay Product (EDP), initially proposed by Horowitz[8] to evaluate trade-offs between circuit level power saving techniques for digital designs, has been applied by Brooks[9] to more heavily, and some would argue more appropriately, weigh delay by squaring or even cubing the delay factor in the calculation.

Cameron, in a poster presented at SC04 and in later papers, suggests a weighted approach where the delay factor can be weighted based on the priority of performance. We maintain the decision of how to weight performance versus power is largely a policy decision. These metrics can be useful in evaluating individual application efficiency on a platform. We contrast performance versus energy use in various ways including EDP and weighted EDP to as fairly as possible present our data.

In [10] the authors focus on exploiting parallel processing inefficiencies to achieve savings in power with little performance impact. Their motivation and goals are very similar to ours in this sense. The authors use micro-benchmarks on what appears to be the aforementioned NEMO 16 node cluster or a close equivalent. With micro-benchmarks, the authors achieve a respectable 25% energy savings with only a 2% performance impact. Our results for some real scientific applications are better which could suggest that potential savings increase with scale.

Ge, Feng and Cameron (et al.) provide an important contribution with their research. Our motivations, and the importance of collecting real measurements are shared. We feel we have improved on their work significantly by analyzing real applications on a large scale High Performance Computing platform (the majority of their work is accomplished analyzing synthetic benchmarks). Our instrumentation allows us to collect data from thousands of nodes, in-situ, at a higher frequency without interrupting the operating system to obtain the data. Additionally, we are able to tune not only the CPU frequency but also analyze the effects of network bandwidth tuning on both power and performance.

Li et al. in [11] models hybrid MPI/OpenMP from a performance and energy perspective examining both dynamic concurrency throttling (DCT) and DVS. In [12] Li takes a modeling approach to investigates task aggregation to reduce energy consumption by reducing the number of nodes. Li uses AMG along with some NPB MPI benchmarks, one of the few efforts that use a real HPC applications for their analysis. They seem to suggest that in some cases the benefits of DVS diminish with scale. Our results using AMG seem to differ but they are using a hybrid implementation which is likely quite different. Their work predicts the performance energy tradeoff up to 1024 cores (128 nodes). In both [11] and [12] the System G supercomputer at Virginia Tech is used. System G consists of 324 nodes. The power measurements appear to be done in a similar manner to the NEMO cluster, I assume the predecessor to System G. This work takes a different approach than our research, more model and analytical based versus our empirical approach. The investigation of hybrid approaches is timely considering the likely-hood of more cores per node, both homogeneous and heterogeneous, in the future. Our work would provide an excellent validation platform for their research to a much higher scale than they have currently investigated.

Hsu and Kremer in [13] investigate a compiler based approach to leverage DVS in efforts to reduce energy consumption during times where the CPU can be slowed without greatly affecting performance, for example during memory stalls. Hsu's later work [6] investigates a run-time approach which does not require source code modification, compiler based or otherwise. This work is also orthogonal to our research but shares many of the motivations or our efforts.

The use of performance counters to estimate power efficiency has been researched from a micro

[14, 15] and macro [16] perspective. While estimates based on performance counters have shown to be useful, we have shown that scalable fine-grained measurements can be leveraged at both the micro and macro level to analyze HPC operating system and application power use with and without frequency scaling and while employing network bandwidth tuning. While our approach is orthogonal, clearly, modeling efforts would benefit by combining these approaches if only for validation purposes.

In [17], the authors evaluate various methods of measuring power including cabinet level collection on the Cray XT architecture. The method is clearly much more coarse and their methods proved to not be scalable. We have considered using cabinet level data as a verification of our data collection mechanisms and may leverage this technique, if it can be implemented in a scalable way, in the future. The authors concluded that measuring power at system scale is problematic and while nodal and small scale measurements can be accomplished the scalable, large scale, collection of samples is not feasible. Our work has shown that it is possible to collect scalable, fine granularity, high-frequency power measurements on HPC platforms given the necessary hardware and software infrastructure. Further, by enabling collection from all nodes used in an application (during a single application run) we can observe and quantify the additional power effects of parallel applications, at scale, rather than extrapolate based on nodal measurements. By leveraging what will hopefully become a ubiquitous capability in the future, power analysis of both operating system and applications on HPC systems can be greatly enhanced.

Kodi et al. [18] discusses the ability to tune network bandwidth using techniques similar to DVS (but for network components) to dynamically reconfigure optical interconnects with the goal of increasing energy efficiency. The authors make many of the same assertions we make in this paper. While Kodi focuses on an analysis of the specific optical interconnect technology, we believe we go a considerable distance towards confirming that large benefits can be experienced by using tunable network components on HPC platforms.

We found work proposing DVS on network links as early as 2003 (see Shang et al.[19]) but we are not aware of any HPC platform that currently has this capability. While our approaches are clearly different, Shang uses a modeling approach to project energy savings while applying a history based policy, our motivations are shared. While a history based approach may not be appropriate for the space we are targeting, using the models developed by Shang a generally efficient policy might be discovered. Shangs work illustrates that implementing DVS requires consideration of many trade-offs.

Brightwell et al. [20], [21] and [22] and Pedretti et al. [23] analyze many aspects of network performance but do not evaluate power as part of their experiment. We have found no work that measures energy in-situ at any scale as it relates to network tuning.

While our investigations suggest it is possible, there are significant challenges in developing network hardware to support the equivalent of DVS. Our goal is to continue to provide strong motivation for tunable network technologies.

Probably the greatest difference and contribution of our work is the sheer scale of of our experiments involving the largest set of real HPC scientific applications. Our work is clearly focused

on empirical analysis. As we have discussed, while this area of research is still in relative infancy, important related work has been done. Our work removes the necessity of extrapolation to large scale and demonstrates true scale effects for scientific applications. Our work involving tuning the network bandwidth and evaluating the impact with these same metrics currently has little related research to compare.

Chapter 2

Research Platform

The only difference between men and boys is the cost of their toys. – Author Unknown

Hardware Architecture

The experiments conducted as part of this research were all accomplished on some variant of the Cray XT architecture. To our knowledge, this is the only platform that exposes the ability to measure current draw, in situ, as described in Chapter 3. Both the idle cycle and frequency scaling experiments require specific operating system modifications to the Catamount light-weight kernel[24]. Catamount support is currently limited to the Cray XT architecture. The Cray XT architecture also affords the rare ability to tune performance parameters of other components. We exploit this capability to tune network bandwidth while measuring the effect on application energy in Chapter 6. The following sections will describe specific test platforms and configurations in more detail as they pertain to our research. It is important to note, obtaining dedicated time on production HPC platforms is difficult and very expensive.

Red Storm

Red Storm was developed jointly by Cray Inc., and Sandia National Laboratories. Red Storm was the first of the Cray XT architecture line. The Cray XT architecture has been installed at numerous government and commercial sites including Oak Ridge National Laboratories. Red Storm is currently a heterogeneous architecture containing both dual and quad core processors. We utilize both variants in our experiments. All nodes, dual and quad, are connected via a Seastar 2.1 network interface controller/router (in short, Seastar NIC) in a modified mesh (mesh in X and Y directions and a torus in the Z direction).

Dual Core Nodes The network bandwidth experiments described in Chapter 6 were accomplished on the dual core (XT3) partition of Red Storm. The XT3 Partition contains 3,360 AMD 64 bit 2.4 GHz dual-core processors with 4GB of DDR memory (2 GB per compute core). Each XT3 node is connected to the network via a Seastar NIC. The ability to manipulate the network

bandwidth of the platform is equivalent on both the XT3 and XT4 partitions. The primary driver of using the XT3 partition for the network bandwidth experiments was simply the availability of this partition. We conducted our idle experiments on both the dual and quad core partitions of Red Storm.

Quad Core Nodes RedStorm's XT4 partition contains AMD 64 bit 2.2 GHz quad-core processors with 8 GB of DDR2 memory (2 GB per compute core). Red Storm has 6,240 quad-core compute nodes each connected to the network via a Seastar NIC. The frequency scaling experiments described in this paper were conducted solely on the quad-core processors of either Jaguar or Red Storm due to the advanced power management (APM) architectural requirements. How we exploit APM features will be discussed in Chapter 6. Some of the applications used in our experiments are export controlled which prevented us from using Jaguar solely for the frequency scaling experiments. Since the architectures and software stacks used were identical, we simply maximized our use of each platform based on application requirements.

Jaguar

Use of Jaguar was granted through the Department of Energy's Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program. Jaguar, located at Oak Ridge Leadership Computing Facility (OLCF), was used for a portion of the frequency scaling experiments outlined in Chapter 6. We specifically employed the XT4 partition of Jaguar since it was both easier to gain dedicated access to and the architecture supported Catamount with much less up front effort than the XT5 partition would have required. Dedicated access was necessary for a number of reasons, primarily driven by our requirement to run Catamount (no longer a Cray supported operating system for the XT architecture). The XT4 partition of Jaguar contains 7,832 64 bit quad-core AMD Opteron processors (or nodes). Each core executes at 2.2 GHz and has access to 8 GB of DDR2 memory (2 GB per compute core). Each node on Jaguar is connected to the network via a Seastar NIC. The network topology of the Jaguar XT4 partition is a 3D torus[1]. Jaguar's network topology differs somewhat from Red Storm's. These differences are not significant to our experiments and had no affect on our results.

Operating System

Serial number one of the Cray XT architecture employed a light-weight kernel operating system named Catamount. Catamount, at the time, was the latest in the lineage of light-weight operating systems authored, or co-authored, by Sandia National Laboratories. Catamount was basically designed to get out of the way of the application. Important hardware abstractions and memory management are all provided with performance being the primary design consideration. When a parallel application is launched Catamount provides the initial set up for the application, including

[1]The XT4 partition of Jaguar was recently decommissioned.

contiguous memory allocation, and basically suspends itself other than handling interrupt driven tasks such as those from network devices (Seastar NIC). This is a simplistic description but sufficient for our purposes. The basic point is that Catamount is a small deterministic operating system, in contrast with general purpose operating systems such as Linux. While it has proven to be a very successful production operating system it has proved invaluable for operating system research at Sandia National Laboratories.

Our first effort as part of this research was to save energy during idle cycles since early versions of Catamount ignored this as a design consideration and to some extent preceded many of the Advanced Power Management (APM) capabilities found on recent processor architectures. We explored further power efficiencies by leveraging more advanced APM features like frequency scaling. The deterministic nature of Catamount greatly aided in conducting this research. More detail on specific modifications will be included in our coverage as necessary.

Reliability Availability and Serviceability System

Historically, Reliability Availability and Serviceability (RAS) systems were commonly provided by vendors on mainframe class systems. Today RAS systems are mostly unique to very high end custom HPC class architectures (Cray XT/XE and IBM BlueGene L/P and Q for example). Its hard to define a clear line where cluster management systems become RAS systems. Generally, cluster management systems consist of a loose collection of open source utilities that are individually designed for a narrow purpose. They are seldom integrated in any way with each other and are often intrusive to the primary purpose of the platform, computation. RAS systems, in general, are typically more intentionally designed and integrated, often specific to a single architecture (in my opinion one of the failures of RAS system designs historically).

The following are excerpts from the requirements for a recent capability class procurement by the Alliance for Computing at Extreme Scale (ACES), a collaboration between Sandia National Laboratories and Los Alamos Laboratory.

- *To achieve delivery of the maximum continuous system resource availability, the RAS system must be a well engineered, implemented and integrated part of the proposed platform.*
- *There shall be a separate and fully independent and coherent RAS system.*
- *The RAS system shall be a systematic union of software and hardware for the purpose of managing and monitoring all hardware and software components of the system to their individual potential.*
- *Failure of the RAS system (software or hardware) shall not cause a system or job interrupt or necessitate system reboot.*

While this is only a small portion of the requirements that described and specified the RAS system for Cielo[25], it suggests some differentiating characteristics between a generic cluster manage-

ment system and what is considered a RAS system. For our research, one of the most important characteristics is the separation but close integration of the RAS system in relationship to the capability platform. This allows out of band[2] scalable collection of the current and voltage data that is used in all of our experiments. It is very important that our experimental methods do not, or minimally, affect the normal activity of the system. In related works, we describe other research efforts that employ laptops and measure power using the ACPI interface. This causes an operating system interrupt each time a measurement is requested. While the interruption is minimal, at scale this type of measurement could introduce the equivalent of operating system noise. There is no such interruption during our measurements. The separate RAS network additionally allows the collection of these measurements in a scalable manner, which is required since during our experiments we collect measurements at high frequency and fine granularity over thousands of nodes simultaneously. In chapter 7 we will discuss our efforts to accomplish similar or possibly improved collection on commodity components.

[2]Out of band, in this context, means that control and monitoring of the platform, in general, is accomplished without affecting the platform or the software running on the platform. Measuring current and voltage data, for example, does not require an operating system interrupt.

Chapter 3

Measuring Power

To understand the affect we must be able to measure the effect. – Unattributed

Overview

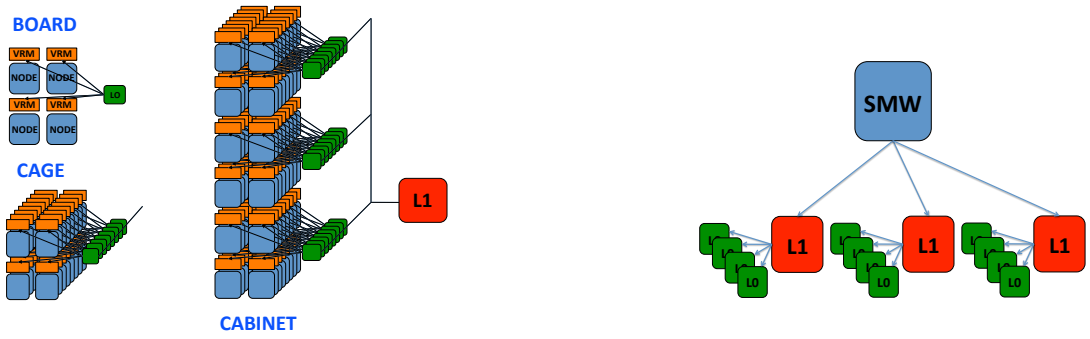
The effort we have expended to measure current draw and voltage are enabling technologies supporting our research. To date there has been no other work published that has been based on such large scale fine grained in-situ measurements of energy on a High Performance Computing (HPC) platform. In this chapter we will discuss both the hardware features, and software development to exploit these features, employed to enable our research.

Hardware

The Cray XT architecture contains a Reliability Availability and Serviceability system (RAS) comprised of both hardware and software with the goal of increasing the reliability of the overall platform. At a very high level, the RAS system is responsible for the control and monitoring of the underlying platform. The separate hardware allocated for the RAS system is intended to ensure the primary purpose of the underlying platform, computation, is affected as little as possible. The distinct but closely integrated nature of the RAS system provides an out-of-band path that allows a large variety of monitoring data to be collected. We will describe how we exploit the existing RAS software infrastructure to meet our goals in the following section.

Unlike typical commodity hardware, the Cray XT3/4/5 node boards provide interfaces that can be exploited to measure component level current draw and voltage. Figure 3.1(a) depicts the logical board, cage and cabinet connectivity of important components of the RAS system involved in collecting current and voltage measurements. Each node board has an embedded processor called an L0 or *level 0* (depicted in green). The L0 has the ability to interface with many on board components. To obtain current and voltage measurements we exploit the L0's i2c[1] connectivity

[1]The i2c is one of several two wire serial bus protocols commonly used for this type of component control and monitoring. SMBus, for example, is another common standard.



((a)) Board, cage and cabinet connectivity

((b)) Hierarchical connectivity

Figure 3.1: Reliability Availability and Serviceability (RAS) Hardware Configuration and hierarchical topology

to the Voltage Regulator Modules (VRM). Note, each node (depicted in blue) has an associated VRM (depicted in orange). The black lines connecting the L0 to each nodes VRM represent the i2c interface.[2] The current and voltage measurements are collected by the L0 on each board. In the Cray XT architecture there are eight boards in a cage. Three cages comprise a cabinet. At the cabinet level there is an additional embedded processor called an L1 or *level 1* (depicted in red). Each of the 24 L0's in a cabinet are connected to the cabinet L1 via Ethernet, also depicted by black lines. The L1 acts as a parent for each of the L0's in a cabinet.

Figure 3.1(b) represents the overall hardware RAS hierarchical topology down to the L0 level. Each cabinet level L1 in the platform connects to the top level System Management Workstation (SMW) (depicted in blue) via Ethernet. Similar to how the L1 acts as a parent for each underlying L0, the SMW acts as a parent for all L1's in the platform. This configuration forms the RAS hardware hierarchy for the Cray XT architecture. This hierarchical configuration provides for sufficient scalability for the control and monitoring of very large platforms. While not infinitely scalable in its current configuration, we have encountered no scalability issues during our experiments.

Software

While the ability to exploit the hardware (collect current and voltage data) is not currently a feature provided by the **C**ray **R**eliability Availability and Serviceability **M**anagement **S**ystem (CRMS), the existing software infrastructure can be leveraged to meet our goals.

The CRMS consists of a number of persistent daemons which communicate in a hierarchical manner to provide a wide range of control and monitoring capabilities. We have augmented the base CRMS software with a *probing* daemon that runs on each L0 and a single *coalescence* daemon that runs on the top level SMW. The *probing* daemon registers a callback with the event loop

[2]In reality, the connectivity is more complicated, involving a number of PIC micro-controllers and Field Programmable Gate Arrays (FPGA). For the purposes of our discussion this is sufficient detail.

executing in the main L0 daemon process (part of the standard CRMS) to interrogate the VRM at a specific bus:device location (corresponding to each individual node or processor socket). In the standard CRMS, the L0 daemon processes communicate to their parent L1 daemon process (also of the standard CRMS, executing on the cabinet level L1) through an event router daemon (additionally part of the standard CRMS). In turn, each L1 communicates to the top-level SMW through an event router daemon. The results of a series of timed probes, requested by the probing daemon, are combined and communicated through the standard event router daemons to the *coalescence* daemon on the SMW, which outputs the results.

The output is a formatted flat file with timestamped hexadecimal current and voltage values for each CPU socket monitored (results are per socket not per core). Figure 3.2 depicts a few sample output lines from an actual experiment. The cname (c14-2c0s3), for example, corresponds to the L0 that resides in the cabinet with X coordinate 14 Y coordinate 2 cage 2 slot 3. The 2nd field is the time-stamp relative to the L0 that collected the data. Some challenges were encountered when we collected data from L0's with poorly synchronized date and time. The remaining fields are the current and voltage measurements for this board. The entries omit the values for nodes 2 and 3 in the interest of space. Note, the last field labeled *mezz* represents the Seastar NIC. As you can see in these few samples the current draw is very consistent for the Seastar and does not change in response to load.

```
c14-2c0s2,1300491313,n0=0x17,n0_V=0x4fa,n0_W=0x1d,n1=0x0f,n1_V=0x503,n1_W=0x13, .... ,mezz=0x46
c14-2c0s3,1300491313,n0=0x1a,n0_V=0x500,n0_W=0x21,n1=0x1f,n1_V=0x4fd,n1_W=0x27, .... ,mezz=0x46
c14-2c0s4,1300491313,n0=0x15,n0_V=0x4e3,n0_W=0x1a,n1=0x14,n1_V=0x4e6,n1_W=0x19, .... ,mezz=0x46
```

Figure 3.2: Sample raw current, voltage and calculated wattage output data

By leveraging the existing hardware and software foundation of the CRMS in this way we have been able to achieve a per socket collection granularity at a frequency of up to 100 samples per second[3]. The accuracy of each sample is approximately +/-2 amperes. While the accuracy of the sample is not as precise as we would like, the data remains extremely valuable for comparing deltas which most of our conclusions are based on. This is in contrast with most other platforms where measuring current draw is typically limited to inserting a meter between a power cable and energy source which results in a very coarse measurement capability at best. (It should be noted, we can also obtain current draw for the Network Interface Controller (NIC or Seastar) but we have observed a constant current draw for this device. This information is useful to quantify the total power used but does not vary depending on usage. We use this measurement for the baseline in our network bandwidth experiments. The current draw measurements include memory controller activity but not power used by the memory banks themselves. The granularity and frequency of this sampling capability has enabled us to observe real power usage in new and powerful ways.

We have closely monitored the impact our instrumentation has had on CRMS. Even at 100 samples per second we have seen little impact on the L0. Likewise, no adverse impact on communication between the L0's and L1 controllers, or between the L1's and the SMW has been observed. We have tested this instrumentation on up to 15 Cray XT cabinets (1440 nodes) and have observed no scaling issues. With this said, until tested at larger scale we cannot confirm functionality beyond

[3]The data used in our research is generally at a 1 sample per second frequency, we found little benefit in higher frequency collection for the purposes of this analysis.

what we have reported. It should be noted that this was not intended to be a production implementation. Cray Inc. has recently become interested in replicating our approach and we expect this to be a standard option of the CRMS in the near future.

Post Processing Measurement Data

In all of our experiments, current and voltage measurements are collected, simultaneously, from each node (more specifically each node's VRM) at a frequency of one sample per second over the duration of the experiment. We collect power data from a large subset of the nodes used in each test. To achieve better coverage for each experiment we distribute the nodes we collect from in a uniform manner. This has the added effect of collecting nearly the same proportion of information from each experiment. Our testing has shown that collecting samples from larger numbers of nodes than we have used does not significantly affect the results obtained.

Our post processing begins with ensuring all samples used are from nodes used in that specific experiment. We then synchronize the data samples. The resulting file is used as input for a Perl script which accomplishes the vast majority of post processing automatically. The post processing script we developed has a wide range of capabilities used for analyzing input data for a number of purposes. The following describes a typical analysis.

In Figure 3.2 notice that a single line of data contains information for all four nodes on a board (and the Seastar or mezz). Each line is first parsed into individual node data and stored in a data-structure which allows us great flexibility with post-processing options. Each line also contains a time-stamp. Each data sample for each node is stored with an associated time-stamp value. As part of our typical post processing we use the trapezoidal rule to integrate these values over time, which approximates the energy used over the duration of the application. This value is expressed in Joules and calculated for each individual node. In addition to this calculation we also produce a graph for each node over the duration of the data sample. We typically output graphs (using gnuplot) of the absolute measurement values using Watts on the X axis and time on the Y axis. We can alternatively plot the measured values relative to the measured idle current. A number of output graph types and formats can be specified. We find for displaying energy used over the duration of an application run the gnuplot filled-curves format works well.

In addition to calculating an energy value and graph for each node represented in the data file we also produce a statistics file. The statistics file contains the energy for each node along with an number of statistics. Our analysis includes the total energy (sum), the average or mean, median, mode and the coefficient of variation (CV). This analysis allows us to process huge amounts of data while ensuring the results are valid. We rely primarily on the CV to ensure our measurements are dependable since the CV is expressed as a percentage, independent of the magnitude of the data. For the purposes of our analysis we are primarily concerned with the differences between complete samples or deltas. We have found in this and previous experiments that the deltas are very reliable and provide a solid foundation for comparison. We use this process to produce the data and graphs that appear throughout this document.

Chapter 4

Applications

The applications used in our research were primarily selected based on their importance to the three Department of Energy National Nuclear Security Administration (NNSA) nuclear weapons laboratories (Sandia, Los Alamos and Lawrence Livermore). As part of the procurement of Cielo, (DOE/NNSA's most recent HPC capability platform (2010)) each laboratory in the Tri-Lab complex specified two production scientific computing applications that would be used in the acceptance phase of the procurement of Cielo. These applications are herein referred to as the *6X* applications (due to the requirement they, on average, must perform six times faster on Cielo, not that there are six applications). The 6X applications include; SAGE, CTH, AMG2006, xNOBEL, UMT and Charon. In addition to the 6X applications we used LAMMPS, another production DOE application, and two synthetic benchmarks, HPL and Pallas. The following sections provide short descriptions of each application used.

High Performance Computing Applications

SAGE[26, 27] SAIC's Adaptive Grid Eulerian hydro-code, is a multi-dimensional, multi-material, Eulerian hydrodynamics code with adaptive mesh refinement that uses second-order accurate numerical techniques. SAGE represents a large class of production applications at Los Alamos National Laboratory. Both strong and weak scaling inputs were used in our experiments.

CTH[28] is a multi-material, large deformation, strong shock wave, solid mechanics code developed at Sandia National Laboratories. It has models for multi-phase, elastic viscoplastic, porous and explosive materials. Three dimensional rectangular meshes; two-dimensional rectangular, and cylindrical meshes; and one-dimensional rectilinear, cylindrical, and spherical meshes are available. CTH has adaptive mesh refinement and uses second-order accurate numerical methods to reduce dispersion and dissipation and produce accurate, efficient results. For our experiments the test problem used was a 3-D shaped charge simulation discretized to a rectangular mesh.

AMG2006[29], developed at the Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, AMG is a parallel algebraic multi-grid solver for linear systems arising from problems on unstructured grids. Based on Hypre[30] library functionality, the benchmark, configured for weak scaling on a logical three dimensional processor grid $px \times py \times pz$, solves the Laplace equations on a global grid of dimension $(px \times 220) \times (py \times 220) \times (pz \times 220)$.

xNOBEL[31], developed at Los Alamos Laboratories, is a one, two, or three dimensional multi-material Eulerian hydrodynamics code used for solving a variety of high deformation flow of materials problems . The problem used for this study was the *sc301p* shape charge problem in two dimensions in a weakly-scaled configuration.

UMT[32] is a 3D, deterministic, multigroup, photon transport code for unstructured meshes. The transport code solves the first-order form of the steady-state Boltzmann transport equation.

Charon[33], developed at Sandia National Laboratories, is a semiconductor device simulation code. Charon employs the drift-diffusion model, a coupled system of nonlinear partial differential equations that relate the electric potential to the electron and hole concentrations. A fully-implicit, fully-coupled solution approach is utilized where Newton's method is used to linearize the discretized equations and a multigrid preconditioned iterative solver is used for the sparse linear systems. Charon uses the solvers from the Sandia National Laboratories Trilinos[34] project. The problem used for this study is a 2D steady-state drift-diffusion simulation for a bipolar junction transistor with approximately 31,000 degrees of freedom per MPI rank.

LAMMPS[35, 36] is a classical molecular dynamics code, and an acronym for **L**arge scale **A**tomic/**M**olecular **M**assively **P**arallel **S**imulator. LAMMPS has potentials for soft materials (biomolecules, polymers) and solid state materials (metals, semiconductors) and coarse grained or mesoscopic systems. It can be used to model atoms or, more generically, as a parallel particle simulator at the atomic, meso, or continuum scale. When run in parallel, LAMMPS uses message passing techniques and a spatial decomposition of the simulation domain.

Synthetic Benchmarks

Highly Parallel Computing Benchmark **(HPL)**[37] is the third benchmark in the Linpack Benchmark Report, used as the benchmark for the bi-annual Top500 report[38]. While the value of the benchmark in measuring how a system will perform for *real* applications can be debated, its pervasiveness is unquestionable. The benchmark solves a random dense linear system in double precision arithmetic on a distributed memory system. The HPL benchmark is well understood and recognized as a compute intensive application.

Pallas[39], now called the **I**ntel **M**PI **B**enchmark (IMB), successor to Pallas GmbH, is a suite of benchmarks designed to measure the performance of a wide range of important MPI routines. Pallas is communication intensive.

Chapter 5

Affecting Power During Idle Cycles

Motivation and Goals

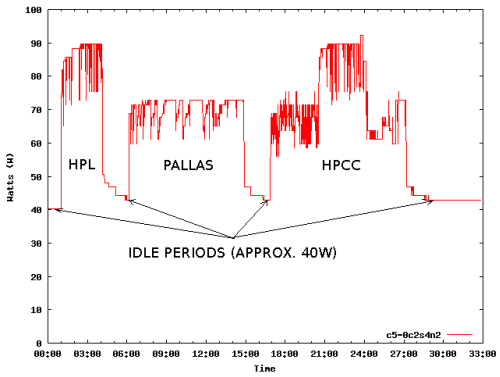
The Linux™community has long been concerned with power saving measures particularly in the mobile computing sector. Linux has been quick to leverage architectural features of microprocessors to reduce power consumption during idle cycles (and under load as we will discuss in Chapter 6). The HPC community makes great use of Linux on many of their platforms but light weight kernels (LWK's) are often used to deliver the maximum amount of performance at extreme scale (Red Storm and Blue Gene, for example). To achieve greater performance at scale, LWK's often have a selective feature set when compared to general purpose operating systems like Linux. As a result, LWK's are a prime area for investigating opportunities for power savings as long as performance is not affected. In the area of idle power usage Linux serves as an established benchmark. Our first goal will be to match or beat the idle current draw of Linux.

Operating System Modifications

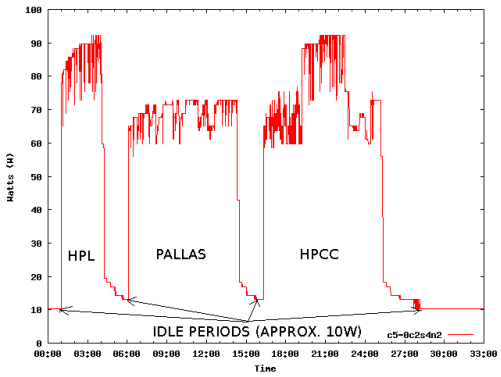
We leveraged the measuring capabilities described in Chapter 3 to examine the current draw of the Catamount LWK. Our initial findings were not surprising. As we suspected, but could not previously quantify or directly observe, idle cycles were consuming current as Catamount busily awaits new work.

One of the advantages of most LWK's (Catamount is not an exception) is the relative simplicity of the operating system. The last two versions of Catamount (Catamount Virtual Node (CVN) and Catamount N-Way (CNW))[1] have supported multi-core sockets. The architecture of Catamount is such that there are only two regions the operating system enters during idle cycles. We first addressed the region where cores greater than 0 (in a zero based numbering scheme) enter during idle. (We will call core 0 the *master* core and cores greater than 0 *slave* cores.) We modified Catamount to individually halt *slave* cores when idle and awaken immediately when signaled by the *master* core. The result was a significant savings in current draw.

[1]The name Catamount will generally be used throughout this document unless the more specific names CVN and CNW are necessary to point out and important distinction. Further specific information about Catamount can be found in [24]



((a)) Compute Node Linux (CNL)



((b)) Catamount Virtual Node (CVN)

Figure 5.1: IDLE Power Comparison between Compute Node Linux and Catamount on an early dual core architecture

As the number of cores per socket increase the savings will likely increase on capability platforms. Capability class applications are typically memory and/or communication bound. Adding more cores, generally, provides less than proportional and applications often run on one or two of the available cores. It should be emphasized that each *slave* core enters and returns from the halt state independently, resulting in very granular control on multi and many core architectures. After these very positive results, we then modified the region of the operating system the master core enters during idle. While the master core is interrupted on every timer tick (the slaves are not) we still observed significant additional power savings during idle periods.[2] Note, if only one core is used during an application run it executes on the master core.

Results and Analysis

Idle Power: Before and After

Figure 5.1(a) depicts measurements obtained running three applications (HPL[37], PALLAS[39] and HPCC[40]) on a Dual Core AMD Opteron Processor[3] using Compute Node Linux (CNL). Figure 5.1(b), in contrast, illustrates the results obtained when executing the same three applications on the same CPU using Catamount. (In our testing we typically compare results using the same exact hardware in an attempt to limit variability of measured results.)

The most noticeable difference between the two graphs is the idle power wattage. CNL uses approximately 40W when idle in contrast to Catamount which uses approximately 10W (prior to our operating system modifications Catamount used approximately 60W). Later results obtained

[2]Timer ticks are configured to occur 10 times per second on Catamount

[3]AMD Opteron 280 AMD Dual-Core Opteron 2.4GHz 2M Cache Socket 940 OSA280FAA6CB

on quad core AMD Opteron[4] sockets showed nearly identical idle power wattage measurements for both CNL and Catamount[5](delta within accuracy of measurement). On this particular dual core architecture the instructions `MONITOR` and `MWAIT` are not supported. Both instructions are supported on the quad core architecture used in subsequent testing. Linux can be configured to poll, halt or use MONITOR/MWAIT during idle. It is possible that what we are observing in Figure 5.1(a) is a polling loop which in Linux is optimized to conserve power. Later observations on the quad core architecture were likely the result of CNL exploiting MONITOR/MWAIT. Regardless, these results are intended to show the ability to observe and contrast. These measurements have demonstrated our first goal of equaling the idle power savings of Linux.

These results also provided our first look at what we have termed Application Power Signatures (see Section 5). Each application has a characteristic signature. While small differences in the signature can be observed, even when running the same application on a different operating system the signature is easily recognized.

A few more subtle points should be made. Without the ability to examine power usage at this level we could only guess that Catamount was inefficient during idle periods and we could not quantify the efficiency. Additionally, we would not have been able to easily measure the effect of our modifications and determine, definitively, when or if we reached our goal. Likewise, when using CNL, we could make the assumption that CNL benefits from power saving features of Linux but without this capability we would not have recognized the difference in power use between the two CPU architectures.

Using the information obtained we can make some simple calculations for a hypothetical system. For the purposes of this calculation we make the following assumptions: a 13,000 node (dual core) system, 80% utilized, 20% idle, ignoring downtime. The idle node hours for this system over a year would be:

$$(13000\,nodes * 0.2) * (365\,days/year * 24\,hours/day) = 22.776 * 10^6\,node\,hours/year \quad (5.1)$$

If we calculate the idle Kilo-Watt hours saved based on 50W per node (the delta between the pre-modified Catamount idle wattage and the modified Catamount idle wattage) we get:

$$(22.776 * 10^6\,node\,hours/year * 50\,Watts/node) \div 1000 = 1.1388 * 10^6\,KW\,hours/year \quad (5.2)$$

Assuming 10 cents per Kilo-Watt hour based on Department of Energy averages for 2008[41] we can calculate real dollar savings for this hypothetical system.

[4]AMD Opteron Budapest 2.2 GHz socket AM2

[5]CVN was enhanced to support more than two cores, the resulting Catamount version was named CNW. Unless otherwise specified all results shown after Figure 5.1(b) were obtained running on CNW

$$(1.1388 * 10^6\, KW\, hours/year * 10\, cents/KW\, hour) \div 100\, cents/dollar = \mathbf{113880\ dollars/year} \quad (5.3)$$

For a capability system using a figure of 80% utilization in the way we have characterized is probably optimistic. Capability systems are typically intended to support one to several large applications at one time which tends to drive the total resource utilization numbers down. Additionally, this calculation does not consider idle cores resulting from applications that use less than the maximum cores available per node (as previously discussed). In the case of dual core sockets half of the resource could remain idle (in power saving mode) when the system is considered to be 100% utilized. In the case of quad core sockets three fourths of the resource could potentially remain idle.

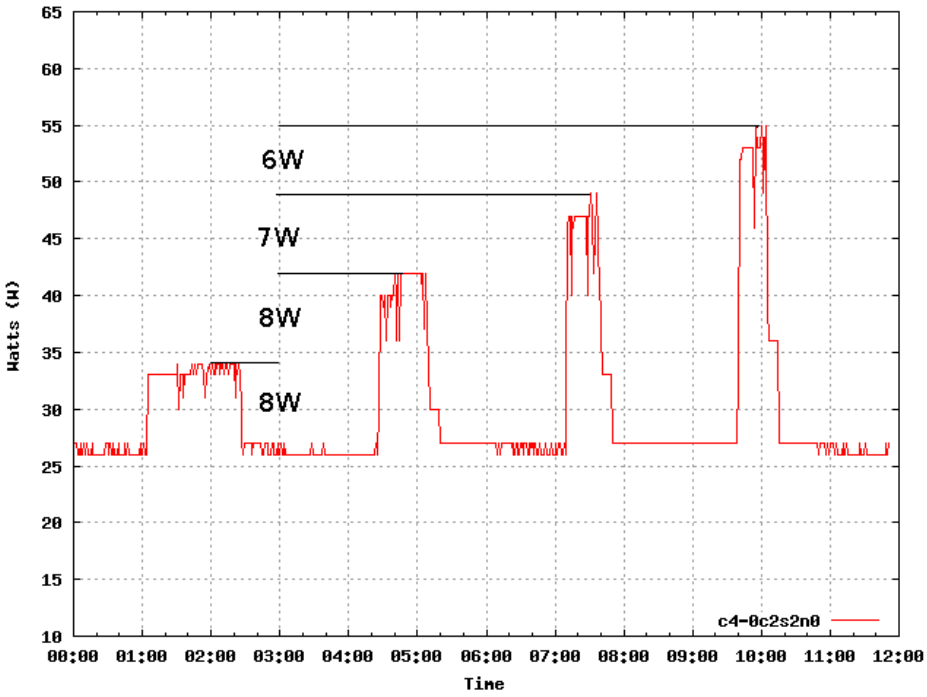


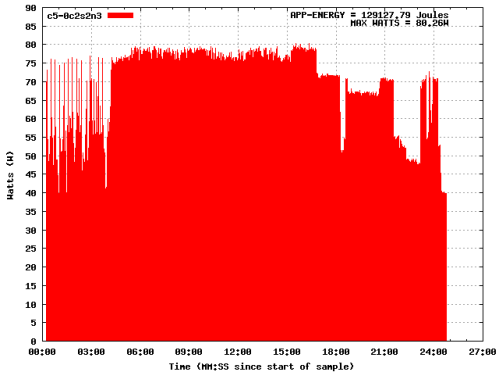
Figure 5.2: Catamount N-Way Per Core Power Utilization

Figure 5.2 illustrates incremental power usage on a quad core socket when a short HPL job is executed on one, two, three and four cores of a quad core node. Even though our measurements are on a per node basis we can see the incremental rise in power usage when additional cores are enlisted. These results provide both a nice illustration of per core savings and a confirmation that our operating system modifications properly handle per core idle states.

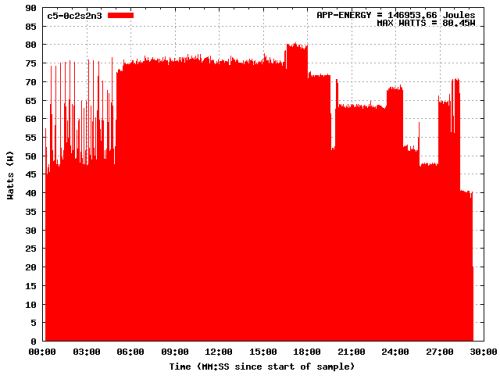
In addition, we have not considered the 30-40% additional power savings as a result of not having to remove the additional heat generated by higher idle wattages. If we include cooling our cost savings range from between $148,044 and $159,432 per year. By exploiting power saving measures, as we have illustrated, significant savings can be realized by targeting idle cores alone.

Application Power Signatures

Application Power Signature is a term we have applied to the measured power usage of an application over the duration of that application. The term signature is used since each application exhibits a repeatable and somewhat distinct shape when graphed. We have found that a user knowledgeable of the application flow can easily distinguish phases of the application simply by viewing the signature. While simply graphing the resulting data can be useful, we have extended this by calculating the energy used over the duration of the application. We call this application energy. To calculate this metric we simply calculate the area under the curve. To accomplish this we enhanced our post processing code to approximate the definite integral using the trapezoidal rule. The following graphs (Figures 5.3(a) and 5.3(b)) depict the data collected while running HPCC on Catamount and CNL. HPCC was executed using the same input file on the same physical hardware. Each run used 16 processors (four nodes, four cores per node).



((a)) HPCC on Catamount



((b)) HPCC on CNL

Figure 5.3: Application Power Signature and Application Energy of HPCC executed on Catamount and Compute Node Linux

In the upper right hand corner of each graph is the energy used by the application (on a single node, all four cores). Again, notice the similarity of the signatures regardless of the underlying operating system. In this case HPCC finished more quickly on Catamount than CNL. HPCC and other applications have been shown to execute more quickly on Catamount[42]. It is not surprising that an application that takes longer to execute, given similar power draw during execution, will consume more power. In this case HPCC ran 16% faster on Catamount. The amount of energy used by HPCC is 13% less using Catamount than CNL. We also tested HPCC on quad core nodes using two cores per node (HPCC ran 15% faster on Catamount and used 13% less power) and on dual core nodes using two cores per node (HPCC ran 10% faster on Catamount and used 10% less power). The salient point is that performance is not only important in reducing the run time of an application but also in increasing the power efficiency of that application. Additionally, without the ability to examine real power use at this granularity the power efficiency of an application could not be as precisely quantified.

Power and Noise

Operating system interference, also referred to as noise or jitter, is caused by asynchronous interruption of the application by the system software on the node. This interruption can occur for a variety of reasons from the periodic timer "tick" commonly used by many commodity operating systems to keep track of time to the scheduling points used to replace the currently running process with another task or kernel daemon.

The detrimental side effects of operating system interference on HPC systems have been known and studied, primarily qualitatively, for nearly two decades [43, 2]. Previous investigations have suggested the global performance cost of noise is due to the variance in the time it takes processes to participate in collective operations, such as `MPI_Allreduce`. LWK's, like Catamount, are essentially noise-less in comparison to general-purpose operating systems like Linux. Previous work has shown that operating system noise can have substantial impact on the performance of HPC applications [44]. In addition, this work shows the impact varies by application, some showing relatively no impact in noisy environments while others exhibit exponential slowdowns. While many aspects of the impact of noise on run time performance are well understood, the impact of noise in terms of power usage is not. Specifically, we set out to answer if power usage in noisy environments scales linearly (or otherwise) with the increase in application run time.

To evaluate the impact of noise we use the kernel-level noise injection framework built into the Catamount LWK [44]. This framework provides the ability to direct the operating system to inject various per-job noise patterns during application execution. The available parameters for generating the noise patten include: the frequency of the noise (in Hz), the duration of each individual noise event (in us), the set of participating nodes, and a randomization method for noise patterns across nodes (not employed for this analysis). The noise is generated (simulated) using a timer interrupt on core 0 of the participating nodes. When the interrupt is generated, Catamount interrupts the application and spins in a tight busy-wait loop for the specified duration. The purpose of separately specifying the frequency and duration of each noise event is to simulate various types of noise that occur on general purpose operating systems. Catamount provides an ideal environment for these studies due to its extremely low native noise signature.

In the following analysis we focused on a single application (SAGE). We chose SAGE based on our initial studies and the previous analysis done in [44]. Applications like SAGE have the potential to be significantly impacted by noise and any proportional increase in energy.

Table 5.1 is a representative sample of our results.

We injected a number of different noise patterns varying the frequency and duration of the noise. The Noise percentage (column one) is determined using the following calculation.

$$((Frequency(Hz) * Duration(us)) \div (1 * 10^6)) * 100 \tag{5.4}$$

The frequency of the noise (column two) is how often a noise event occurs. The duration

Table 5.1: Power impact of Noise

Noise	Freq	Duration	Diff Runtime	Diff App Energy (AVG)
2.5%	10Hz	2500us	4.0%	4.0 %
1%	10Hz	1000us	1.7%	1.9%
2.5%	100Hz	250us	2.6%	2.5%
2.5%	1000Hz	25us	2.6%	2.5%
1%	1000Hz	10us	0.1%	0.1%
10%	**10Hz**	**10000us**	**21.6%**	**21.0%**

(column three) is how long each noise event lasts. The difference in runtime is shown in column four and is relative to the runtime of the application with no noise injected. Likewise, the difference in application energy (column five) is relative to the energy used by the application without noise injected. The results, with the exception of row six, are representative of multiple runs on the same equipment using the same parameters. In addition, we varied the runtime of the application with consistent results. The results were obtained using 16 quad core nodes. The application utilized core 0 only since noise can only be injected on core 0 using this framework. What we observed is that the difference in application energy used by applications when noise is injected is linearly proportional to the difference in runtime. If we normalize the impact of the injected noise, even in the most extreme example (again excluding row six) the impact of noise on both the runtime and the application energy is approximately 1.5%. We found these results to be very consistent. We repeated our tests at a larger scale (48 nodes, again utilizing only core 0) and observed results consistent with Table 5.1. In an effort to simulate effects seen at larger scale we introduced a large amount of noise (10%) while running the same application. The results (row six of table 5.1) show a larger impact to both runtime and application energy (approximately 11% when normalized). These results are significant in the fact that they show the same linearly proportional increase in application energy for applications effected by noise. Though Table 5.1 shows small percentage increases in runtime for various noise patters, accompanied by proportional increases in the percentage of energy used, these results were obtained at a relatively small scale. The run time of some applications can increase dramatically at larger scale in noisy environments.

In Figure 5.4, we see the measured slowdown of POP, CTH, and SAGE, at scale. In this figure the Y-axis is the global accumulation of noise for the application. We compute this global accumulation by taking the slowdown of the app in a noisy environment versus a baseline with no OS noise and subtract the amount of locally injected noise. For example, if we inject a 2.5% net processor noise signature, as we did for this figure, and measure a 20% slowdown the global accumulation of noise would be 17.5%. We see in this figure that with only 2.5% net processor noise injected the slowdown for POP exceeds 1200% at scale, and therefore we can project a proportional increase of 1200% application energy at scale. The inset of Figure 5.4 also shows considerable slowdowns for SAGE and CTH due to noise. While not as dramatic as POP, the additional impact on application energy projected by our analysis is proportionally as significant. Further analysis will need to be accomplished to verify that these results are truly representative at scale.

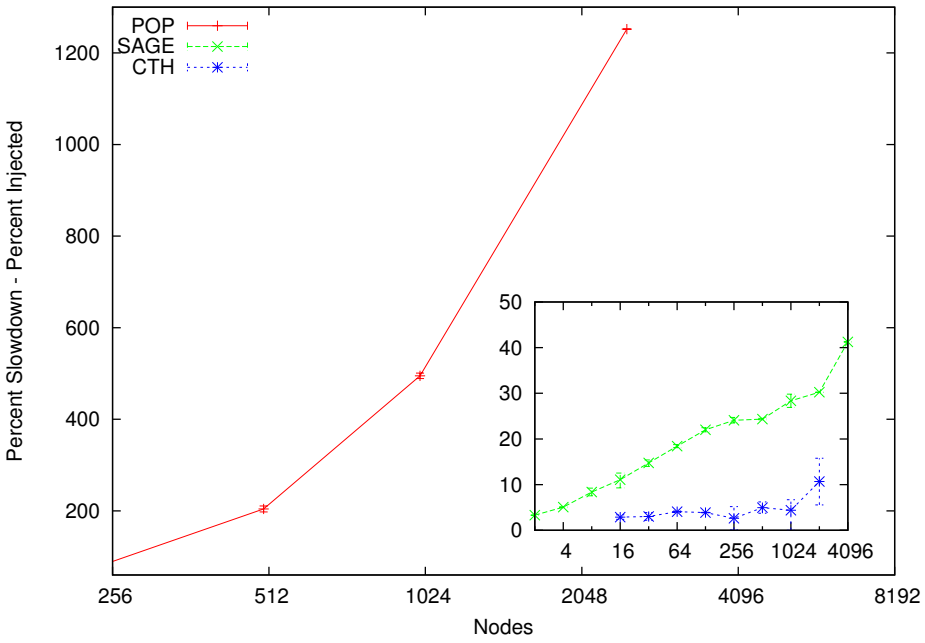


Figure 5.4: Slowdown at Scale

Chapter 6

Tuning Power During Run-time

Motivation and Goals

In our previous experiment we focused on reducing power by exploiting idle cycles. In the following experiments we research opportunities to reduce the energy use of a running application without affecting performance. As mentioned previously, determining what is and is not an acceptable trade-off between energy and performance is somewhat subjective. The motivation of this research is to show that significant energy savings can be achieved by tuning architectural components on a per application basis. In this experiment we focus on CPU frequency scaling during an application run.

Typical approaches employed by operating systems, such as Linux, while efficient for single server or laptop implementations, have proven to be detrimental when used at scale causing the equivalent of operating system jitter[2]. For this reason, most if not all sites that deploy clusters at medium to large scale disable these features while running applications (some sites enable these features during idle cycles). It is clear to us that techniques designed for laptop energy efficiency are not directly applicable to large scale High Performance Computing (HPC) platforms. In our CPU frequency scaling experiments, we take a more deterministic approach ensuring all cores participating in an application are executing at the target frequency in lock step. We call this static frequency modification. In this experiment we contrast the runtime impact with the energy savings on a per application basis. This analysis focuses on CPU energy. In a separate experiment we contrast runtime impact with total system energy while tuning the network interface, again on a per application basis. We feel both approaches provide valuable information.

Feng reports in [3] that the CPU is the largest single component consumer of energy on a node. While Feng's analysis is for a commodity board that contains disk and other components we can perform a similar analysis on the hardware used for our experiment. A Cray XT architecture node board contains only CPU, memory and a Network Interface Controller [1]. Using a value of 20W for memory and 25W for the NIC we can use our measured CPU energy data over the duration of the application to calculate an average Watts for the CPU for each individual application. On our platform the CPU ranges from 44-57% of the total node power. Clearly, since the CPU is the largest consumer of power it is productive to analyze it both in isolation and as a component of

[1] We do not include the small amount of power used by the embedded controller since it would be amortized across the four nodes present on the board and be largely insignificant for this purpose.

total system power. Additionally, we analyze CPU data in isolation because this information is more directly applicable in comparison to other commodity based platforms.

Static CPU Frequency Tuning

Operating System Modifications

To accomplish our goals, we made a small number of targeted modifications to Catamount[2]. First, it is necessary to interrogate chip architecture capabilities. Generally, we are determining if Advanced Power Management (APM) is supported. More specifically, we interrogate the CPU to determine if hardware P-state[3] control is supported. Changing P-states requires writing to P-state related Memory Status Registers (MSR). If APM is not supported, writing to P-state MSRs is fatal. Even if APM is enabled, however, only a single P-state is required to be defined. In addition, even if multiple P-states (up to 5) are defined, they may have identical definitions. This is typically not the case but enforces the importance of closely interrogating specific hardware capabilities. From this point forward we assume APM is supported and multiple P-states are defined, at least two of which define different operating frequencies.

Currently, our method of frequency scaling is limited to frequencies defined in the P-state table, although most processors support frequency stepping in 100MHz increments. The impetus of changing frequency is ultimately to lower the input voltage to the processor. Power is proportional to the frequency, capacitance and voltage squared. By this definition the largest impact to power can be obtained by lowering input voltage. Both the processor and the infrastructure must support dynamic voltage transitions for us to take advantage of this potential power savings. While it is unlikely that future architectures will support independent per core power planes there will likely be multiple power domains per processor chip. Understanding how these power planes are partitioned will be important to achieve maximum energy savings. On the test platforms used for our experiments all cores were required to be in the same *higher*[3] P-state before a lower input voltage could be achieved. Basically, if one core is operating at a higher frequency (which requires a higher input voltage) the input voltage to the processor remains at the voltage necessary to support the highest active frequency (current lowest active core P-state or current operating frequency (COF)). While describing the subtleties of serial and parallel voltage planes is beyond the scope of this paper they are very important architectural details and cannot be overlooked.

At a very early stage in the boot process we collect the default P-state and supported P-states of

[2]Additional detailed information specific to the AMD architecture family discussed here can be found in the BIOS and Kernel Developers guide (BKDG)[45]

[3]From BKDG: P-states are operational performance states (states in which the processor is executing instructions, running software) characterized by a unique frequency and voltage. The processor supports up to 5 P-states called P-states 0 through 4 or P0 though P4. P0 is the highest power, highest performance P-state; each ascending P-state number represents a lower-power, lower performance P-state than the prior P-state number. As P-state numbers increase, the operating frequency and voltage for a given P-state must be less than or equal to the frequency and voltage of the prior P-state. At least one enabled P-state (P0) is specified for all processors.

P-state	CPU frequency	Input Voltage
0	2.2 GHz	1.2 V
1	2.0 GHz	1.2 V
2	1.7 GHz	1.15 V
3	1.4 GHz	1.075 V
4	1.1 GHz	1.05 V

Table 6.1: Test Platform P-states, CPU frequencies and Input Voltages

each core. This information is stored and used by a trap function that we added to handle a variety of P-state related functionality. Since changing P-states is a privileged call (writing to MSRs) the ability to change P-states was added in two parts; an operating system trap and a user level library interface. The trap implements query functionality to determine what P-states are available, what P-state the core is presently in and of course the ability to transition from the current P-state to an alternate supported P-state. The trap also reports the final P-state achieved and in debug mode the number of nanoseconds the P-state transition took. The amount of time necessary to transition between P-states is not important for the experiments covered in this paper since we accomplish a single static change prior to application execution. Transition time becomes a critical consideration when more dynamic methods of CPU frequency scaling are employed. We discuss our efforts in this area in Chapter 7.

Table 6.1 lists the supported P-states, corresponding CPU frequencies and required input voltages from Red Storm. Note that the default P-state on Red Storm is P-state 0. Testing was conducted on P-states 0, 2, 3 and 4 on Red Storm. On Jaguar the default P-state is P-state 1. Testing on Jaguar was conducted using P-states 1, 2, 3 and 4. We observed some inconsistency in the reported P-state vids reported. Since we directly measure both the current and the voltage these inconsistencies would not affect our data.

The frequency in MHz at the end of each entry is calculated using the CPU *fid* (core frequency ID) and the CPU *did* (core divisor ID) (see BKDG[45] for addition information). The *vid* (core voltage ID) specifies the necessary input voltage to support the operating frequency of each P-state. Notice also the *nbvid* (north-bridge voltage ID) in each case is the same as the vid. On our test platform the input voltage for both the CPU and the north-bridge must be the same.

The input voltage, represented in hexadecimal, can be found in the BKDG for the processor family. The 10h family supports both a parallel and a serial voltage interface. The systems used in our testing use the serial voltage interface infrastructure. A vid of 0x1c corresponds to 1.2 volts[4]. By selecting P2 we can lower the input voltage to vid 0x20 (1.15 volts). In this example notice the vids for P0 and P1 are identical. Using P1 offers little advantage to us since the operating frequency (fid) is lowered with no accompanying reduction in voltage.

Our current implementation of this trap has proved to be extremely stable. We transition between P-states in a step-wise fashion. If, for example, P-state four is requested and the core is

[4]To determine the voltage the hexadecimal value must first be converted to its binary representation then referenced in the appropriate table for either serial or parallel power planes in the BKDG[45]

currently in P-state zero we transition through P-states zero through four one by one. We have found this approach to be more reliable than directly requesting cores to transition to specific P-states. Additionally, the transition time does not impact our results at least in the case of static P-state operation.

To confirm that our P-state transitions are successful we employed our ability to monitor both current draw and voltage on a per socket granularity. After requesting a change in P-state we can observe a corresponding drop in input voltage. Note, we have noticed in some cases, even when the transition to a new lower voltage P-state has been confirmed, a drop in input voltage does not occur. While this is sub-optimal it has the effect of understating our results since we calculate energy based on measured voltages. In other words, if all voltages behaved ideally, our energy savings would be greater than we have reported.

We observed more issues while running on Red Storm than on Jaguar. Further investigation led us to discover that while Jaguar has very recent PIC revisions, Red Storm's PIC revisions are quite dated. This is partially due to the age of Red Storm. Since Red Storm was serial number one of this architecture, PIC updates were expected to be infrequent therefore remote update capability was not incorporated into the platform. Benefiting from the lessons learned from earlier installations, the hardware used on Jaguar does support remote PIC updates and therefore it is much less intrusive to keep PIC firmware up to date. Our experiments proved to run much better on Jaguar likely due to the up-to-date PIC firmware which affects many of the hardware characteristics we manipulate.

Library Interface

Since changing P-states (changing COF) is a privileged operation, the trap is accessed through a variety of functions provided by a user level library. While a single trap function implements all of the frequency change functionality, for ease of use and clarity we have implemented a library function interface to exploit each capability individually.

- **cpu_pstates(void)** - Returns detailed processor P-state information
- **cpu_freq_step(P-state)** - Request a P-state transition (up or down)
- **cpu_freq_default(void)** - Returns default processor P-state

In this experiment, we quantify the affects of CPU frequency modification. We refer to this experiment as *static* frequency modification. Prior to executing the user application we first execute a *control* application. The control application simply changes the CPU's COF by changing the CPU's P-state to the desired level (cpu_freq_step(P-state)). The control application is launched on every core of each CPU that will be used in the test application. Note, while convenient for our testing, a separate control application would not be required in a production environment.
P-state changes are accessible from any portion of the software stack using this library interface.

Following execution of the control application we execute the HPC application under test on the same nodes. The HPC application will run at a lower frequency defined by the P-state selected

by the control program. During the execution of the HPC application we collect data for both current draw and voltage at one second intervals. Once the HPC application is completed we return all nodes to the default P-state or select a new P-state for a subsequent experiment with cpu_freq_step(P-state).

The trap and library interface was designed to support both static and dynamic frequency scaling. Initially, we anticipated that it would be necessary to change frequency often during application execution to achieve an acceptable trade-off between performance and energy. In testing our modifications, we discovered that large benefits exist for static frequency scaling. Static frequency scaling has many benefits including simplicity and stability. We continue to pursue dynamic frequency scaling. Our efforts regarding dynamic CPU frequency modification will be touched on in Chapter 7.

Results and Analysis: CPU Frequency Tuning

We will refer to tables A.1 and A.2 and figure 6.1 in our discussions. Increases in run-time or energy percentage in tables A.1 and A.2 are indicated by positive numbers, negative values are indicated by parenthesized numbers (all relative to the baseline values listed). In all cases great care was taken to use the same nodes for each application execution. Figures 6.1(a) and 6.1(b) were created by overlaying the energy results from an individual node for P-states 1-4 running the specified application. Tables A.1 and A.2 are the result of separate experiments. We show the tables side by side for ease of comparison and analysis.

Our frequency scaling experiments were conducted during five separate dedicated systems times. Four of the experiments were conducted on Jaguar during eight to twelve hour sessions. The final experiments were conducted on Red Storm during a three day dedicated system time. Over this period we conducted a range of experiments using real production scientific applications and synthetic benchmarks listed and described in Chapter 4. As can be seen in table A.1 we were able to test some applications in all available P-states. Others exhibited clear results in early testing and did not warrant further experiments and in some cases we were simply unable to obtain results at higher P-states (lower frequencies) due primarily to hardware issues (primarily PIC revision issues as previously described). However, as can be seen in table A.1 our results are nonetheless extensive.

Decreasing CPU frequency, in general, will slow active computation. If applications were solely gated by computation this approach would be entirely detrimental. However, applications exhibit a range of characteristics. In this experiment, we altered CPU frequency and measured the impact on energy and run-time (other platform parameters are left unchanged). We begin our analysis with a discussion of the extremes represented by two synthetic benchmarks, HPL and Pallas. Note, for all experiments we will contrast both run-time and energy to the baseline runs conducted at P-states 0 or 1 (depending on the platform used) and report the contrast as percent increase or decrease from the baseline value in table A.1. For the baseline runs we record the execution time in seconds (s) and the energy used in Joules (J). Our CPU frequency experiments focused on the affect that CPU frequency modifications had on CPU energy alone. In [3] the

authors evaluate CPU power as a percentage of total system power for both idle cycles and during application load. Even at idle the CPU accounts for 14% of the total power draw according to their measurements. More significant to our study there measurements suggest that during load the CPU accounts for more than a third (35%) of the total node power draw. Later we take a broader look at total system energy but clearly measuring the impact of the CPU alone is important. We feel evaluating CPU power in isolation and evaluating total system power provide important insights. **Might consider a pie chart for the two extremes of CPU power vs total node power, related work indicates the CPU is the largest single contributor to the total power of a node**

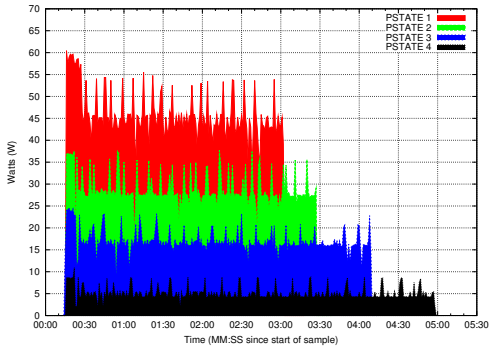
HPL is largely a compute intensive application. We chose HPL to demonstrate an application that would potentially be highly impacted by reducing CPU frequency. Our HPL results were as expected. In table A.1 we see that a change to P-state 2 causes a 21.1% increase in run-time and a 26.4% decrease in energy used. This would likely not be an acceptable trade-off for a real application unless the priority was energy savings.

In contrast to HPL, Pallas (IMB) is a highly communication intensive benchmark. Pallas was chosen to demonstrate an application that would be potentially less affected by reductions in CPU frequency. Again, as expected, Pallas demonstrates only a 2.30% increase in run-time and a 43.6% reduction in energy used when run in P-state 2. This would likely be a favorable trade-off for any application. Given the results from these synthetic benchmarks we expect our real applications will fall somewhere in between these extremes. We will address the *real* applications in table order.

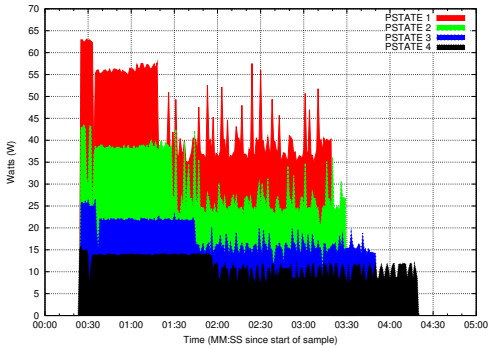
We were able to obtain results for AMG2006 at P-states 1-4 at a scale of 6K cores. At P-state 2 we observed an increase in run-time of 7.47% accompanied by an energy savings of 32.0%. Note, we used the longest of three run-times in each case for our final measurements. AMG2006 had a short run-time and we found the shortest run-time in P-state 2 was actually faster than the longest run-time in P-state 1. For this reason we are confident in saying that AMG2006 could benefit from a reduction in frequency to, at a minimum, P-state 2. The trade-off at P-state 3 is not as clear. The runtime impact is proportionally greater than the energy savings at P-states 3 and 4. We do note while P-state 4 exhibits a significant hit in run-time we measured the largest savings in energy we recorded in our experiments. Depending on policies and/or priorities AMG2006 might be able to take advantage of any of the available P-states to produce significant savings in energy.

LAMMPS (tested at 16K cores), in contrast to AMG2006, does not display a clear win when run at lower frequencies. Our results for P-state 2 show a 16.3% increase in run-time and a 22.9% decrease in energy. Not a clear win but in some cases this might be an acceptable trade-off. The results for P-states 3 and 4 demonstrate a very significant hit in run-time. While it is our opinion that increases in run-time of this magnitude are not acceptable LAMMPS does, however, show a correspondingly large savings in energy at P-states 3 and 4. Again, this may be acceptable in some circumstances where energy consumption is the primary consideration or policy decisions enforce energy limitations.

Figures 6.1(a) and 6.1(b) graphically depict each of the four executions of AMG2006 and LAMMPS at P-states 1-4. The shaded area under each curve represents the energy used over the duration of the application. Figure 6.1(b) clearly depicts the positive run-time vs. energy trade-off (for AMG2006) indicated in table A.1 especially between P-states 1 and 2. In contrast,



((a)) LAMMPS P-states 1-4



((b)) AMG P-states 1-4

Figure 6.1: Application Energy Signatures of LAMMPS and AMG2006 run at P-states 1-4

more dramatic increases in run-time can be seen in figure 6.1(a) for LAMMPS. While AMG2006 showed a favorable trade-off between run-time and energy when run at lower frequencies there might be even more benefit to obtain. Notice the compute intensive phase of AMG2006 early in the application execution. If we were to maintain high CPU frequency during this phase but transition to lower frequency for the remainder of the application it is likely that the benefits would be even greater. While LAMMPS did not show a clear win when lowering CPU frequency, notice the very regular compute phases throughout the entire application execution (indicated by peaks in the graph). In Chapter 7 we will discuss how these application characteristics, once understood, could be leveraged to obtain power savings even in applications that do not present a clear choice when we statically modify the CPU frequency.

We obtained results for SAGE using a weak scaling problem at two different scales (4K and 16K cores). In both cases, a small increase in run-time (0.402% at larger scale and 3.86 % at smaller scale) is observed with a very significant reduction in energy (39.5% at large scale and 38.9% at small scale). We were able to obtain results for a 4k core run of SAGE at Pstate 3. The impact on run-time almost doubled but remains low while some additional energy savings were recorded. Based on our observations, it is possible that the 16k core SAGE would have demonstrated a favorable trade-off at P-state 3.

CTH was executed at P-states 0, 2 and 3 at a scale of 16K cores. Similar to LAMMPS, there is no clear win with CTH when the CPU frequency is lowered. Also, like LAMMPS, CTH has very regular compute and communication phases. LAMMPS and CTH will likely be the targets of our future work in dynamic CPU frequency scaling at large scale. **Might provide an MPI profile picture of CTH?**

We obtained results for xNOBEL at 6K cores at P-states 0, 2 and 3. Our results indicate that xNOBEL, like AMG2006, is a good candidate for CPU frequency reduction, even using this static method. Having the ability to tune CPU frequency at large scale for this application would be a clear win.

UMT and Charon behaved in a very similar manner. Since UMT was run at a much larger scale than Charon (16K cores vs. 4K cores) we feel the results obtained for UMT are more meaningful

and more accurately represent what we could expect at large scale. Charon may act differently when run at larger scale but these results indicate that both UMT and Charon are sensitive to CPU frequency changes. It is possible that further analysis will reveal opportunities to dynamically scale frequency during the execution of these applications.

The CPU is only one component that affects application performance. In the following section we will experiment with tuning network bandwidth and observe the resulting performance vs. total system energy trade-offs.

Network Bandwidth Tuning

Enabling Bandwidth Tuning

Our goal was to determine the affect on run-time performance and energy of production scientific applications run at very large scale while tuning the network bandwidth of an otherwise balanced platform[22]. To accomplish network bandwidth scaling we employed two different tunable characteristics of the Cray XT platform. We first tuned the interconnect bandwidth of the Seastar to reduce the network bandwidth in stages to $1/2$ and $1/4^{th}$ of full bandwidth. Since we were unable to reduce network bandwidth further by tuning the Seastar, we tuned the injection bandwidth, effectively reducing the network bandwidth to $1/8^{th}$. This allowed for the most accurate stepwise reduction in network bandwidth we were able to achieve, using this architecture, and a more complete analysis of the affects of network bandwidth tuning.

Modifying the network interconnect bandwidth on the Cray XT3 (or any XT platform using Seastar) requires a fairly simple change to the router configuration file which is consulted (if present) during the routing process of the boot sequence. This unfortunately necessitates a full system reboot for every alteration of the interconnect bandwidth. Typically, all four rails of the Seastar are configured on. This is the default behavior but the number of rails can also be specified in the *rail_enable* field of the router configuration file by specifying a single hex number (representing the four configuration bits[5]). In our experiments, we configured the interconnect bandwidth of the Seastar to effectively tune the network bandwidth to full, 1/2 and $1/4^{th}$.

Since the interconnect bandwidth on the XT architecture is far greater than the injection bandwidth of an individual node, the interconnect bandwidth had to be reduced to $1/2$ before it produced a measurable effect. Its is important to note, the interconnect topology of this platform (Red Storm) is a modified mesh (partial torus). Multiple nodes may route through an individual Seastar depending on communication patterns and where they logically reside in the network topology. For this reason, we limited our experiments to one application execution at a time. This allowed for the nearest estimation of the impact of network bandwidth tuning on an individual application. Running other applications concurrently would be an interesting experiment but would greatly complicate analysis and was beyond the scope of this experiment.

[5] 4 rails = 1111 = 0xF, 3 rails (not used) = 0111 = 0x7, 2 rails = 0011 = 0x3, 1 rail = 0001 - 0x1

Tuning the node network injection bandwidth, to further reduce the overall network bandwidth, requires a bit more effort than tuning the Seastar interconnect required. Fortunately, we have access to the Cray XT bootstrap source code. The portion of the code necessary to modify (*coldstart*) serves an equivalent purpose as the BIOS on a personal computer or server. Early in the power-on sequence, coldstart initializes the HyperTransport (HT) links which join each node to the on board Seastar. The speed of these links determines the injection bandwidth of the node (into the network). It is important to note, each Opteron supports multiple HT links; we only modify the links connecting the node to the Seastar. All other links operate at normal rates, i.e. the links to main memory continue to operate at full speed. In normal operation, the injection bandwidth is determined by the maximum negotiated rate between the node and the Seastar. Similar to modification of the interconnect bandwidth, a reboot is required to configure the injection bandwidth to the desired setting. Normally the links operate at 800 MHz and utilize the full 16 bits of each link. Theoretical full bandwidth is calculated as follows (uni-directional link is full-duplex):

$$800MHz \times 2bits/clock \times 16wires/link \times 1Byte/8bits = \textbf{3.2 GB/sec} \tag{6.1}$$

To achieve $1/8^{th}$ injection bandwidth we configure each link to run at 200 MHz with an 8 bit per link width. The tuned bandwidth is calculated as follows:

$$200MHZ \times 2bits/clock \times 8wires/link \times 1Byte/8bits = \textbf{400 MB/sec} \tag{6.2}$$

We selected this injection bandwidth rate since it further reduced the overall network bandwidth beyond what was possible by reducing the interconnect bandwidth of the Seastar. We should also note that when we reduce the injection bandwidth to effectively reduce network bandwidth to $1/8^{th}$ we only impact the individual node. While all nodes ingress into the network are equally impacted, the network bandwidth available for routing between nodes once on the network, is not reduced. When conducting our experiments, we will use a single set of baseline runs and compare these against identical runs while tuning the network bandwidth using the previously described methods in sequence.

Figure 6.2 depicts the four network bandwidth rates as measured by Pallas PingPong between two dual core nodes, one core per node. As can be seen, the maximum bandwidth observed at $1/2$ using these benchmarks is not $1/2$ of full bandwidth. As previously mentioned, this is due to the larger interconnect bandwidth capacity as it relates to injection bandwidth. Injection bandwidth was not altered other than to achieve the $1/8^{th}$ bandwidth configuration. Below $1/2$ bandwidth, the steps become regular. While not perfect, using the configuration techniques available this is the best approximation that could be achieved. It should be noted that our goal was to test at reduced network bandwidths and measure impact on performance and energy. These configurations clearly achieve our goal and represent a tunable network bandwidth capability.

The following is the sequence of experiments for the network bandwidth study:

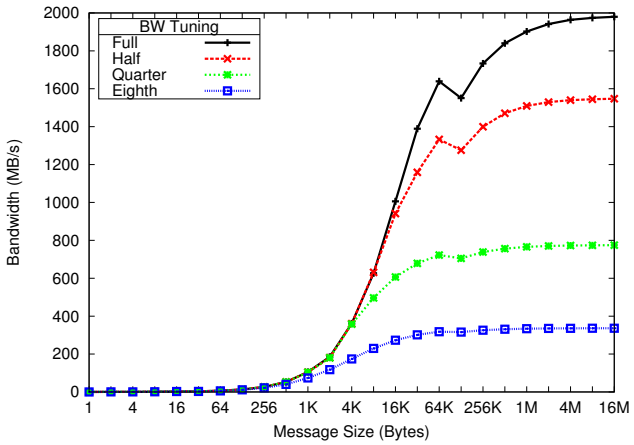


Figure 6.2: Pallas PingPong bandwidth for all levels of network bandwidth tuning

1. Run each application at full interconnect and injection bandwidth to establish a benchmark (run-time and energy use)
2. Reboot into a $1/2$ interconnect bandwidth configuration and run each application
3. Reboot into a $1/4^{th}$ interconnect bandwidth configuration and run each application
4. Reboot into full interconnect bandwidth and $1/8^{th}$ injection bandwidth configuration (to accomplish $1/8^{th}$ network bandwidth) and run each application

For each phase we collected power samples (current draw and voltage) as described in Chapter 3. The scale used for each application, number of nodes and cores, is listed in table A.2. The results will be discussed in the following section. No operating system modifications were necessary for either the interconnect or injection bandwidth experiments.

Results and Analysis: Network Bandwidth Tuning

The following discussion will reference table A.2. Total energy in our table includes the measured energy from the CPU, a measured energy from the Seastar and an estimated energy from the memory subsystem. Since we cannot measure the memory sub-system in isolation we calculate the energy used by this component using a fixed wattage over time. While we are able to measure the current draw of the entire mezzanine (the mezzanine contains four Searstar network chips, one per node), the current draw is constant over time. This is typical of network interface controllers (NIC) since the SerDes do not throttle up and down based on network traffic or on demand in current network chips. Since there are four Seastars in a single mezzanine we multiply the current reading by the input voltage then divide the total by four and use the result as our baseline network power value. For $1/2$, $1/4^{th}$ and $1/8^{th}$ network bandwidth calculations we assume a linear reduction in power which has a proportional affect on total energy. For each energy value we calculate a per node use and multiply by the number of nodes to produce our final value. The calculation is as follows (where E = Energy):

$$(E_{cpu} + E_{network} + E_{memory}) \times \text{number of nodes} = \textbf{Total Energy} \quad (6.3)$$

In our calculations we used 25 W for the full network bandwidth value, 12.5 W for $1/2$, 6.25 W for $1/4^{th}$ and 3.125 W for $1/8^{th}$ network bandwidth. We used 20 watts for the memory value in all calculations primarily to avoid the network energy having a disproportional affect on the total energy calculation. While the E_{CPU} value fluctuated based on the CPU usage of the application and was measured over time, the network and memory values assumed a constant value over time. As mentioned previously, the CPU energy was measured on a large subset of the total nodes involved in the experiment. An average per node energy is calculated based on the samples from all nodes and used as the E_{CPU} value. Again note that a decrease, or savings, in energy or run-time is indicated by a parenthesized value.

Addressing each application in table order (see table A.2) we see the strong and weak scaling versions of SAGE have very similar characteristics. Reducing the network bandwidth by $1/2$ had little affect on the run-time of both SAGE_strong (decreased by 0.593%) and SAGE_weak (increased by increased by 0.609%). In the same test, a significant savings in energy was observed for SAGE_strong (a decrease of 15.3%) and SAGE_weak (decrease of 14.3%). The impact on, or increase in, run-time is larger by more than 8X when the network bandwidth is reduced to $1/4^{th}$ for both SAGE_strong and SAGE_weak. Little additional energy savings were observed for this test. As might be expected, further reductions to $1/8^{th}$ network bandwidth for both strong and weak scaling modes of SAGE produce significant impacts the run-time of SAGE (in excess of 20% in both modes). The accompanied energy savings using $1/8^{th}$ network bandwidth is actually smaller than the $1/4^{th}$ network bandwidth experiment. The difference in both run-time and energy savings between strong and weak scaling at $1/8^{th}$ network bandwidth might be an indicator that additional divergence might been seen at higher scale but we cannot conclusively claim this. Based on this data, reducing network bandwidth by $1/2$, if we could reduce the corresponding energy consumption of the network by half, would be advantageous for this application. Considering the run-time energy trade-off we would likely not choose to reduce the network bandwidth further based on the available data.

CTH was more dramatically affected by changes in the network bandwidth than any other *real* application we tested. Even at $1/2$ bandwidth, CTH experiences a greater percent increase in run-time (9.81%) than is saved by reducing network energy (7.09% decrease in total energy). At $1/4^{th}$ bandwidth, CTH experiences a very large increase in run-time (30.2%) accompanied by an actual increase in energy used of 1.04%. Clearly reducing network bandwidth further is highly detrimental to both run-time and energy as can be seen from $1/8^{th}$ network bandwidth results. Even at this moderately large scale CTH requires a high performance network to execute efficiently.

AMG2006 and xNOBEL, in contrast with CTH, are insensitive to the network bandwidth changes we made from the run-time perspective which yields an opportunity for large savings in energy. Reductions down to $1/8^{th}$ network bandwidth cause virtually no impact in run-time for both AMG2006 and xNOBEL while an 25.9% savings in energy can be achieved for both. (AMG2006 executed 0.931% slower while xNOBEL actually ran slightly faster, 0.375%). We do

note the savings in energy seems to be flattening by the time we reduce network bandwidth to $1/8^{th}$. While further reductions in network bandwidth may or may not increase run-time there is likely little additional energy savings available.

UMT produced similar results to AMG2006 and xNOBEL when the network bandwidth was reduced up to $1/4^{th}$, little to no impact in run-time accompanied by a large energy savings. At $1/8^{th}$ network bandwidth we see different characteristics. UMT has a much higher impact to run-time at $1/8^{th}$ network bandwidth (6.32%) than at $1/4^{th}$ (1.07%) with virtually no additional energy savings (21.7% at $1/4^{th}$ and 21.8% at $1/8^{th}$. We seem to have found the limit of network bandwidth tuning that should be applied to UMT at least at this scale and on this platform. We should note that UMT was run at a smaller scale relative to the other applications. It is possible that at larger scale our results might differ.

Charon showed small, but increasing, impact on run-time as we reduced network bandwidth. At this scale it is clear that we could reduce the network bandwidth down to $1/4^{th}$ with probably an acceptable impact in run-time (increase of 2.15%) accompanied by a very significant savings in energy (decrease of 20.8)%). Moving from $1/4^{th}$ to $1/8^{th}$ network bandwidth shows some signs of a flattening of energy savings but results are not conclusive. Experiments with Charon at larger scale are also warranted.

Overall, we observed much evidence that a tunable NIC would be highly beneficial, if corresponding energy savings resulted. In all cases but CTH, virtually no impact to run-time would be experienced by tuning the network bandwidth to $1/2$. The result would be significant energy savings with little to no performance impact. In the case of AMG2006, xNOBEL and UMT the network bandwidth could be reduced to $1/4^{th}$ full bandwidth with little run-time impact, allowing for even larger energy savings. Our observations indicate that a tunable NIC would be beneficial but they also indicate a high performance network is critical for some applications. An ability to tune the NIC, similar to how frequency is tunable on a CPU, would be an important characteristic on next generation Exascale platforms.

It should be stressed that our data is representative of a single application running at a time. One of the reasons the interconnect bandwidth of the Seastar was designed to be greater than the injection bandwidth of a single node is that communications on networks, like the ones used on Red Storm and Jaguar, are not point to point. Often, many hops are required for a messages to travel from source to destination. Having a greater interconnect bandwidth is essential for a platform that supports a range of applications, often sharing the interconnect bandwidth. The ability to tune this component could not be exploited without considering the possible impact on other applications running on the platform, at least for network topologies like meshes and 3D-toruses. Network topologies with fewer hops on average could benefit more easily from a tunable network since less consideration would be necessary regarding the impact on other applications co-existing on the platform.

Energy Delay Product

In this section we will analyze our results using a range of fused metrics including Energy Delay Product (EDP). Many people find it useful to have a single metric to analyze data or make policy decisions. The following discussion will reference figures B.1 and B.2.

The graphs in figures B.1 and B.2 were produced using the same base data used to create tables A.1 and A.2. In all graphs included in figure B.1, the *Runtime* curve is produced by normalizing the runtime measured for each individual application at every P-state tested to the baseline run measured at the default P-state (P0 or P1). The lower X axis lists test points by P-state, the upper X axis lists test points by CPU frequency. The *Energy* curve is produced identically to the *Runtime* curve using measured CPU energy instead of measured runtime. Three EDP curves are present using the following equation:

$$E * T^w \; - \; where: \; E = Energy, T = Runtime and w = 1, 2 or 3 \tag{6.4}$$

We include all three curves to represent how the metric might differ depending on the weight given to time or performance. Weighting the time factor in the EDP equation seems in-line with the existing priorities of HPC. As previously discussed, these priorities might change in the near future.

The HPL results nicely show how weighting runtime (performance) moves the curve upward, indicating a less favorable trade-off as performance is more highly prioritized. Squaring the delay produces an EDP greater than one which would typically be interpreted as detrimental. It should be noted that the unweighted EDP curve trends below one. Using this metric we might choose to run HPL at a lower P-state. This alone should be an indication that if performance is a priority the unweighted EDP metric might not be appropriate. The Pallas graph, however, shows that even if the delay is cubed this metric indicates a benefit running at P2.

While the results for HPL and Pallas are mostly in-line with our previous observations, some of the real application results could be interpreted differently. Even based on the EDP cubed metric, AMG2006, for example, appears to benefit when executed at any P-state including P4. This was not our feeling based on separate runtime and energy differences listed in table A.1. Our basic impression was that a 39.1% hit in runtime would not be acceptable. Considering the percentage hit in runtime in isolation 39.1% does sound extreme, but considering the actual runtime value the EDP metric exposes a dimension that we might have ignored. The runtime for AMG2006 is very short in our tests. Recall we noted that the fastest runs in P-state 2 actually took less time than the slowest runs measured in P-state 1. Since the runtime for AMG2006 is so short, a 39.1% hit in runtime only increases the runtime by approximately 68 seconds. When we are dealing with very short runtimes even cubing the delay in the EDP equation might not be enough.

If we use the cubed EDP as a metric for the remaining applications our analysis closely resembles our initial analysis based on the separate energy and runtime differences. We would conclude

that lowering the CPU frequency for CTH, UMT and Charon is detrimental, while SAGE and xNOBEL are less sensitive to CPU frequency changes.

The graphs in figure B.2 represent our runtime and total energy measurements. The *Runtime* and *Energy* curves are normalized in the same manner as described for figure B.1 with the exception that we measure total energy as described in equation 6.3. EDP is calculated using equation 6.4 using total energy. The X axis lists the steps of network bandwidth reduction. The EDP curves again generally represent our previous analysis. SAGE is not sensitive to our initial network bandwidth reduction to 1/2 but quickly trends negative as we reduce bandwidth further. CTH is very sensitive to all network bandwidth changes. AMG2006, xNOBEL, UMT and Charon in contrast are generally insensitive to network bandwidth changes. Recall, however, that we made some finer judgments based on our analysis of the separate energy and runtime metrics listed in table B.2. In the case of UMT we listed diminishing returns as we decreased network bandwidth. We can see the curves trending upward as we approach $1/8^{th}$ network bandwidth but this seemed easier to identify when analyzing the tabularized data. Regardless, this method of analyzing data can help us quickly get an impression of the trends. If further analysis is warranted we can always view the raw data. We can also conclude that at this point for HPC, the EDP cubed equation is most appropriate.

Chapter 7

Conclusions

Future Work

Our general trajectory for future work is to continue and expand our analysis in hopes of uncovering additional opportunities for increasing energy efficiency at large scale. Our interests are focused primarily on a system approach, although there may be opportunities as we approach Exascale to accomplish node level studies. Exactly how we will achieve Exascale is still unclear, but it is clear is that there will be more node level processing power and it will be in the form of more cores (type and balance of cores and magnitude of *more* is not clear at this point in time). Extracting the most efficient performance at the node level will certainly be part of the system level picture.

Dynamic Frequency Tuning

The experiments involving frequency tuning in this report have leveraged what we have termed *static* frequency tuning. Some applications did not show a clear benefit as a result of static frequency tuning or network bandwidth tuning (also accomplished statically). CTH, for example, displayed a significant runtime hit when we statically modified the frequency or the network bandwidth (tables A.1 and A.2). The question remains, are there ways of increasing the energy efficiency of applications that do not show positive results for static tuning?

Figure 7.1 exposes a possible opportunity for savings if a dynamic approach to frequency, or network, tuning is applied. In figure 7.1 the black lines represent point to point communication, green lines are send, blue are receive, red synchronization, pink reduce and gray areas are periods of computation.

Our intent is to avail the application of the greatest CPU frequency available during computation periods and tune the frequency appropriately during periods of communication. What appropriate tuning is will be determined by our research and as we have already seen we suspect will be somewhat application specific. While we are not currently capable of dynamically tuning network bandwidth we can use the data we have collected to develop a model that will help us predict the benefits of dynamic network tuning.

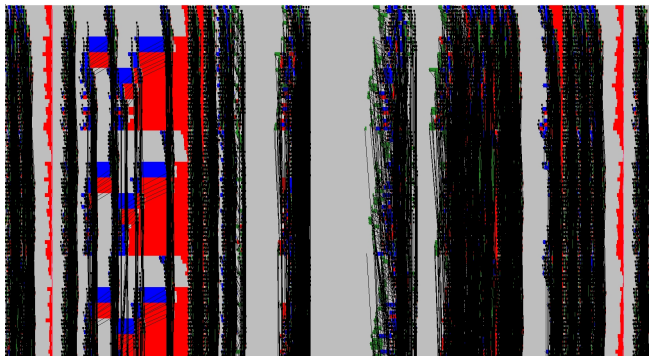


Figure 7.1: CTH MPI Profile, 1 time step on 128 cores

In chapter 6 we pointed out similar opportunities with AMG2006 and LAMMPS. While our results showed that AMG does benefit from static frequency tuning (and network bandwidth tuning) there are portions where it seems that providing the maximum CPU frequency available might be beneficial. In figure 6.1(b) we observe a period of very intense computation for approximately the first third of the application runtime. We also observe additional periods of computational intensity throughout the remainder of the execution. If we dynamically tuned the CPU frequency, and or the network bandwidth, appropriately during these periods we could achieve even greater application energy efficiency. While LAMMPS, like CTH, did not exhibit a clear with with CPU frequency tuning we observe in figure 6.1(a) that LAMMPS displays a very regular period of computational intensity. Like CTH, we may be able to show improvements using dynamic frequency tuning where static frequency tuning was not clearly productive.

There are a number of additional considerations that must be weighed when applying dynamic frequency tuning. Changing frequency based on P-states and more granular CPU frequency tuning takes time (latency). We instrumented our implementation and found that transitions take 10's of microseconds. We have not optimized for transition latency and we are currently only measuring the latency within the operating system trap from the time that the MSR is written until the transition is completed (or reported unsuccessful). On the architecture used for our experiments we use the Time Step Counter (TSC) which for the AMD family 10h processor is invariant. Any implementation which targets dynamic frequency tuning would necessarily have to take great efforts to optimize for transition latency. Much of our future work is planned to be done using Linux and we will be tied to existing methods good or bad. Any future work that involves Kitten will afford us more control over areas that we feel need to be optimized. This area of investigation can benefit from both modeling and experimentation. Once we find the practical bounds of latency overhead of we can determine what duration of computation, or communication, within an application could benefit from dynamic frequency changes.

PMBUS

Summation

NOTES: our next steps: Enable Cielo to accomplish further application tests, scaling studies etc. Small and large scale study on changing frequency during application investigation of new architectural capabilities and how they can be leveraged, or turned off if detrimental more towards real or near real time analysis of measurement data. PMBUS and commodity systems Reducing within an application, MPI profiles indicate regular compute/communication phases Wave Division Multiplex for NICS and turning off lanes Impact of shared links, reduced by networks will smaller radix.

This page intentionally left blank.

Appendices

This page intentionally left blank.

Appendix A

Full Page Tables

Table A.1: Experiment 1: CPU Frequency Scaling: Run-time and CPU Energy %Difference vs. Baseline

		Baseline Frequency		P-2 - 1.7 GHz %Diff		P-3 - 1.4 GHz %Diff		P-4 - 1.1 GHz %Diff	
	Nodes/Cores	Run-time (s)	Energy (J)	Run-time	Energy	Run-time	Energy	Run-time	Energy
HPL	6000/24000	1571	4.49×10^8	21.1	(26.4)				
Pallas	1024/1024	6816	1.72×10^8	2.30	(43.6)				
AMG2006	1536/6144	174	9.49×10^6	7.47	(32.0)	18.4	(57.1)	39.1	(78.0)
LAMMPS	4096/16384	172	2.79×10^7	16.3	(22.9)	36.0	(48.4)	69.8	(72.2)
SAGE	4096/16384	249	4.85×10^7	0.402	(39.5)				
(weak)	1024/4096	337	1.51×10^7	3.86	(38.9)	7.72	(49.9)		
CTH	4096/16384	1753	3.60×10^8	14.4	(28.2)	29.0	(38.9)		
xNOBEL	1536/6144	542	4.96×10^7	6.09	(35.5)	11.8	(50.3)		
UMT	4096/16384	1831	3.48×10^8	18.0	(26.5)				
Charon	1024/4096	879	4.47×10^7	19.1	(27.8)				

Table A.2: Experiment 2: Network Bandwidth: Run-time and Total Energy %Difference vs. Baseline

		Baseline Bandwidth (BW)		$1/2$ BW %Diff		$1/4^{th}$ BW %Diff		$1/8^{th}$ BW %Diff	
	Nodes/Cores	Run-time (s)	Energy (J)	Run-time	Energy	Run-time	Energy	Run-time	Energy
SAGE_strong	2048/4096	337	5.79×10^7	(0.593)	(15.3)	8.90	(15.5)	20.2	(11.4)
SAGE_weak	2048/4096	328	5.64×10^7	0.609	(14.3)	8.23	(15.8)	22.6	(9.63)
CTH	2048/4096	1519	2.58×10^8	9.81	(7.09)	30.2	1.04	40.4	3.50
AMG2006	2048/4096	859	1.45×10^7	(0.815)	(15.8)	(0.116)	(22.7)	0.931	(25.9)
xNOBEL	1536/3072	533	7.01×10^7	(0.938)	(15.4)	(0.375)	(22.2)	(0.375)	(25.9)
UMT	512/1024	838	3.57×10^7	0.357	(14.7)	1.07	(21.7)	6.32	(21.8)
Charon	1024/2048	1162	9.96×10^7	1.55	(13.7)	2.15	(20.8)	2.67	(24.5)

Appendix B

Full Page Figures

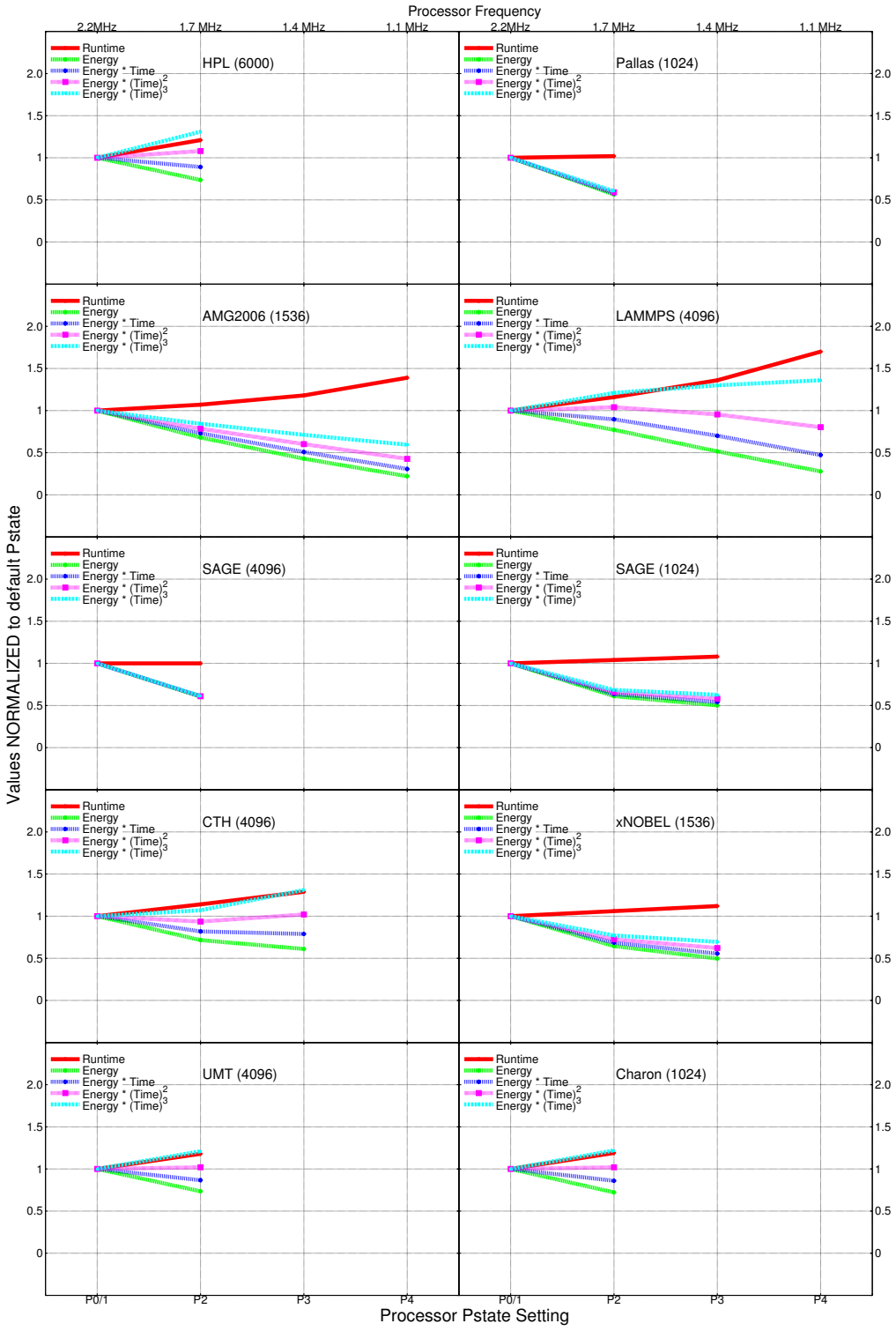


Figure B.1: Experiment 1:Normalized Energy, Run-time and $E * T^w$ where w=1,2, or 3

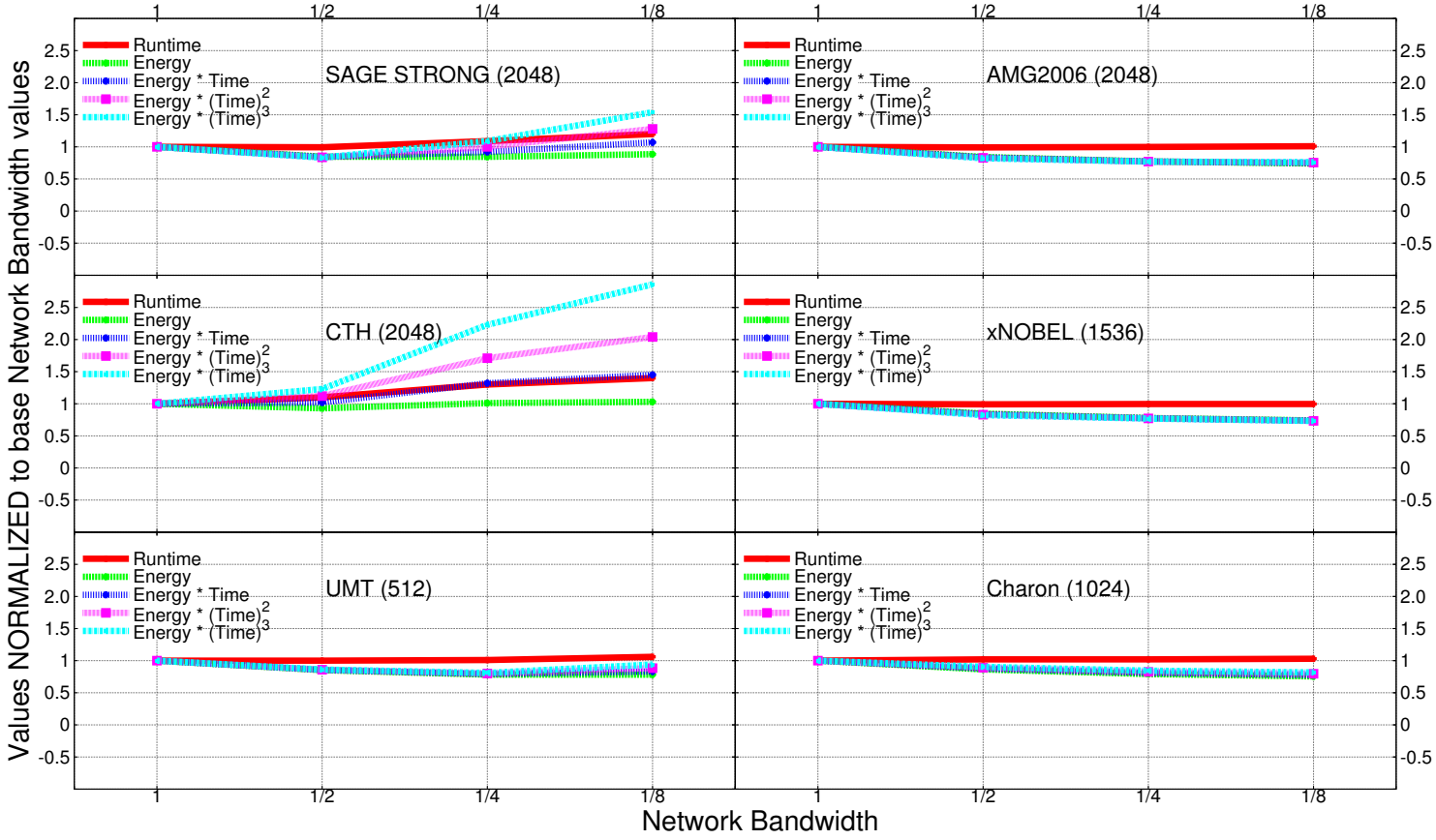


Figure B.2: Experiment 2:Normalized Total Energy, Run-time and $E * T^w$ where w=1,2, or 3

This page intentionally left blank.

References

[1] A. Zavanella and A. Milazzo, "Predictability of bulk synchronous programs using mpi," in *Parallel and Distributed Processing, 2000. Proceedings. 8th Euromicro Workshop on*, 2000, pp. 118 –123.

[2] F. Petrini, D. Kerbyson, and S. Pakin, "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q," in *Proceedings of SC*, 2003.

[3] X. Feng, R. Ge, and K. W. Cameron, "Power and Energy Profiling on Scientific Applications on Distributed Systems," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium IPDPS*. University of South Carolina, Columbia SC, 2005.

[4] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. Cameron, "Powerpack: Energy profiling and analysis of high-performance systems and applications," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 5, pp. 658 –671, may 2010.

[5] E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in *Proceedings of the 5th USENIX conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2007, pp. 2–2. [Online]. Available: http://portal.acm.org/citation.cfm?id=1267903.1267905

[6] C. hsing Hsu and W. chun Feng, "Power-Aware Run-Time System for High-Performance Computing," in *Conference on High Performance Networking and Computing*, 2005.

[7] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, ser. SC '05. IEEE Computer Society, 2005, pp. 34–.

[8] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-power digital design," in *Low Power Electronics, 1994. Digest of Technical Papers., IEEE Symposium*, oct 1994, pp. 8 –11.

[9] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta, and P. Cook, "Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors," *Micro, IEEE*, vol. 20, no. 6, pp. 26 –44, nov/dec 2000.

[10] R. Ge, X. Feng, and K. Cameron, "Improvement of power-performance efficiency for high-end computing," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, april 2005, p. 8 pp.

[11] D. Li, B. de Supinski, M. Schulz, K. Cameron, and D. Nikolopoulos, "Hybrid mpi/openmp power-aware computing," in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, april 2010, pp. 1 –12.

[12] D. Li, D. Nikolopoulos, K. Cameron, B. de Supinski, and M. Schulz, "Power-aware mpi task aggregation prediction for high-end computing systems," in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, april 2010, pp. 1 –12.

[13] C.-H. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for cpu energy reduction," in *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, ser. PLDI '03. New York, NY, USA: ACM, 2003, pp. 38–48. [Online]. Available: http://doi.acm.org/10.1145/781131.781137

[14] F. Bellosa, "The Benefits of Event-Driven Energy Accounting in Power-Sensitive Systems." in *ACM SIGOPS European Workshop*, September 2000.

[15] W. L. Bircher, M. Valluri, J. Law, and L. John, "Runtime Identification of Microprocessor Energy Saving Opportunities." in *International Symposium on Low Power Electronics and Design*, August 2005, pp. 275–280.

[16] W. L. Bircher and L. K. John, "Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events," in *International Symposium on Performance Analysis of Systems & Software*. University of Texas at Austin, April 2007.

[17] S. Kamil, J. Shalf, and E. Strohmaier, "Power Efficiency in High Performance Computing," in *IEEE International Symposium on Parallel and Distributed Processing*, 2008.

[18] A. Kodi and A. Louri, "Performance adaptive power-aware reconfigurable optical interconnects for high-performance computing (hpc) systems," in *Supercomputing, 2007. SC '07.*, 2007, pp. 1 –12.

[19] L. Shang, L.-S. Peh, and N. K. Jha, "Dynamic voltage scaling with links for power optimization of interconnection networks," in *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, ser. HPCA '03. IEEE Computer Society, 2003.

[20] R. Brightwell, B. Barrett, K. Hemmert, and K. Underwood, "Challenges for high-performance networking for exascale computing," in *Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on*, 2010, pp. 1 –6.

[21] R. Brightwell, K. D. Underwood, C. Vaughan, and J. Stevenson, "Performance evaluation of the red storm dual-core upgrade," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 2, pp. 175–190, 2010.

[22] R. Brightwell, K. Predretti, K. Underwood, and T. Hudson, "Seastar interconnect: Balanced bandwidth for scalable performance," *IEEE Micro*, vol. 26, no. 3, pp. 41 –57, 2006.

[23] K. T. Pedretti, R. Brightwell, D. Doerfler, K. S. Hemmert, and J. H. Laros III, "The impact of injection bandwidth performance on application scalability," March 2011.

[24] S. M. Kelly and R. B. Brightwell, "Software Architecture of the Light Weight Kernel, Catamount," in *Cray User Group*, May 2005.

[25] Sandia National Laboratories and Los Alamos Laboratory. [Online]. Available: http://www.lanl.gov/orgs/hpc/cielo/

[26] R. Weaver and M. Gittings, " Massively Parallel Simulations with DOEś ASCI Supercomputers: An Overview of the Los Alamos Crestone Project ," in *Adaptive Mesh Refinement - Theory and Applications*, ser. Lecture Notes in Computational Science and Engineering. Springer Berlin Heidelberg, 2005, vol. 41, pp. 29–56.

[27] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings, "Predictive performance and scalability modeling of a large-scale application," in *Proceedings of the ACM IEEE conference on Supercomputing, Denver CO*, 2001.

[28] E. S. Hertel, Jr., R. L. Bell, M. G. Elrick, A. V. Farnsworth, G. I. Kerley, J. M. Mcglaun, S. V. Petney, S. A. Silling, P. A. Taylor, and L. Yarrington, "Cth: A software family for multi-dimensional shock physics analysis," in *in Proceedings of the 19th International Symposium on Shock Waves, held at*, 1993, pp. 377–382.

[29] R. D. Falgout, P. S. Vassilevski, Panayot, and S. Vassilevski, "On generalizing the amg framework," *SIAM J. Numer. Anal*, vol. 42, pp. 1669–1693, 2003.

[30] http://acts.nersc.gov/hypre/.

[31] M. Gittings, R. Weaver, M. Clover, T. Betlach, N. Byrne, R. Coker, E. Dendy, R. Hueckstaedt, K. New, W. R. Oakes, D. Ranta, and R. Stefan, "The rage radiation-hydrodynamic code," *Computational Science & Discovery*, vol. 1, no. 1, p. 015005, 2008.

[32] UMT2K https://asc.llnl.gov/computing_resources/purple/archive/benchmarks/umt/umt1.2.readme.html.

[33] P. T. Lin, J. N. Shadid, M. Sala, R. S. Tuminaro, G. L. Hennigan, and R. J. Hoekstra, "Performance of a parallel algebraic multilevel preconditioner for stabilized finite element semiconductor device modeling," *Journal of Computational Physics*, vol. 228, pp. 6250–6267, September 2009.

[34] Sandia National Laboratories. [Online]. Available: http://trilinos.sandia.gov/

[35] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *J. Comput. Phys.*, vol. 117, pp. 1–19, March 1995.

[36] LAMMPS http://lammps.sandia.gov/index.html.

[37] J. Dongarra, J. Bunch, C. Moler, and G. W. Stewart, "High Performance Linpack HPL," 1989.

[38] Top500 http://www.top500.org/.

[39] PALLAS http://www.intel.com/cd/software/products/asmo-na/eng/cluster/mpi/219848.htm.

[40] HPCC. [Online]. Available: http://icl.cs.utk.edu/hpcc/

[41] DOE Energy Statistics. Department of Energy. [Online]. Available: http://www.eia.doe.gov/cneaf/electricity/epm/table5_6_a.html

[42] C. T. Vaughan, J. P. VanDyke, and S. M. Kelly, "Application Performance under Different XT Operating Systems," in *Cray User Group*, May 2008.

[43] R. Zajcew, P. Roy, D. Black, C. Peak, P. Guedes, B. Kemp, J. LoVerso, M. Leibensperger, M. Barnett, F. Rabii, and D. Netterwala, "An OSF/1 UNIX for Massively Parallel Multicomputers," in *Proceedings of the USENIX Technical Conference*, Winter 1993.

[44] K. B. Ferreira, R. Brightwell, and P. G. Bridges, "Characterizing Application Sensitivity to OS Interference Using Kernel-Level Noise Injection," in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (Supercomputing'08)*, November 2008.

[45] BKDG: AMD BIOS and Kernel Developers Guide for AMD Family 10h Processors Rev 3.48. Advanced Micro Devices.

DISTRIBUTION:

1	MS 1319	James A. Ang, 1422
1	MS 1319	Ron Brightwell, 1423
1	MS 1319	James H. Laros III, 1422
1	MS 1319	Kevin Pedretti, 1423
1	MS 1319	Sue Kelly, 1423
1	MS 1319	John Vandyke, 1423
1	MS 1319	Courtenay Vaughan, 1423
1	MS 0899	Technical Library, 9536 (electronic copy)

This page intentionally left blank.

Sandia National Laboratories